THE BEST SELLING COMPUTER PROJECTS MAGAZINE SEPTEMBER 1985

ELECTRONICS & COMPUTING

AN EMAP PUBLICATION

USA $2.95
Germany D6.00
Singapore S$4.95

95p

BUILDING ON THE QL

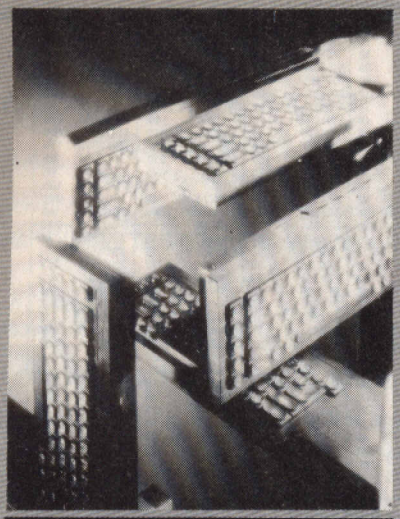
QL MODEMS HEAD-TO-HEAD

QDOS SHELL

DIY CAD SOFTWARE

• HARD TIMES WITH THE ATARI ST • BBC PLOTTER PROCEDURES •

• MAKING MUSIC ON THE AMSTRAD •



ELECTRONICS & COMPUTING Contents

Vol. 5 Issue 9

SOFTWARE

PROJECTS

FEATURES

REVIEWS

PLUS

IMPORTANT ANNOUNCEMENT
See pages 14 and 15 for a sneak preview of Computing Age magazine.

Electronics & Computing Monthly
Priory Court, 30-32 Farringdon Lane,
London, EC1R 3AU

Editorial 01-251-6222
Editor Gary Evans
Deputy Editor William Owen

Advertising 01-251-6222
Advertisement Manager Tony Herman
Advertising Executive Tracey Keighley
Advertising Production Serena Hadley

Production 01-251-6222
Design Boldface
Typesetting & Make-up Time Graphics

Publisher Terry Pratt

Distribution
EMAP National Publications

Published by
EMAP Business and Computer Publications

Printed by
Riverside Press, England

Subscriptions and Back Issues
Subscriptions/
Back Issues telephone 0858 34567

Electronics & Computing Monthly is normally published on the 13th day of each month.

Free public domain software for Amstrad disk users

Owners of an Amstrad disk-based computer system will be interested to hear of a collection of FREE software. What's the catch? You may well ask. Well for once there is no catch.

The software is free because it is in the public domain, ie there is no copyright holder to come down on anyone making copies of the software. 69 such programs have been collected together and a book describing them has been compiled by Davis Rubin Associates. The fact that documentation exists overcomes one of the major problems with PD software in that it is often obtained with little or no instructions.

Programs in the collection cover a wide range of applications from games to business.

In the UK the book plus programs cost £39.95, or £27.95 if the buyer supplies three formatted disks. The book on its own is available for £18.95. Those with even a basic grasp of maths will spot that £27.95 – £18.95 does not equal free. The £9 difference covers the cost of copying the software – if there is a catch this is it.

To order or to request more information contact *Davis Rubin Associates, 1 Canonbourne, Weston sub Edge, Chipping Campden, Glous, GL55 6QH. (Telephone 0386 841181).*

Acorn users alive and well

With all the problems at Acorn over the past six months or so it is not surprising that some pundits were starting to write the company off. In the same breath these self proclaimed computer gurus were alos saying that the BBC micro had had its day and that the various companies supplying the needs of BBC micro owners were in for a very hard time.

Anyone visiting the Acorn User Show in late July would have found little evidence to back up this view of the Acorn/BBC computer. The Show attracted a large number of exhibitors, many with new products on view for the first time. The public turned up in large numbers and quite a few of the visitors had plenty of money to spend. This would seem to dispose of another commonly held view that during the summer months people are not interested in computers and are certainly not spending any money on computers or peripherals.

The the BBC micro is no longer a number one best seller cannot be disputed. Few people would disagree with the view that the Model B is over priced and that the B+ came to the market far too late to stimulate much interest. There are so many BBC micros out there though that it will be many years before interest in the machine begins to fade. Many people who have bought the BBC computer have invested many hundreds of pounds in hardware and software add-ons and it is unlikely that they will sacrifice this considerable investment in order to use a machine that just happens to be flavour of the month.

Electronics and Computing has supported the BBC computer in the past and we shall continue to do so via the pages of *Computing Age*. The first issue of *Computing Age* will carry an update on the original *E&CM* BBC EPROM blower first published in October 1983. This was one of the most popular projects to appear in the magazine and with the improved software to be published next month, this project is bound to have a new lease of life.

In addition we shall be publishing a number of other software hints, tips and projects all of interest to BBC micro owners.

For more details of *Computing Age* see page 14.

GARY EVANS

British company provides OS for Commodore's Amiga computer

The American launch of Commodore's Amiga computer means the end of a very frustrating time for Bristol based software house Metacomco.

The company was called in by Commodore to write the operating system for the new wonder machine when the company originally selected for the job had trouble meeting Commodore's deadlines.

Metacomco's experience with TRIPOS gave it a flying start and a version of this operating software was demonstrated on the Amiga within one month.

The secrecy surrounding the Amiga project meant that Metacomco could only respond with the time honoured 'no comment' to the increasing number of curious press enquiries.

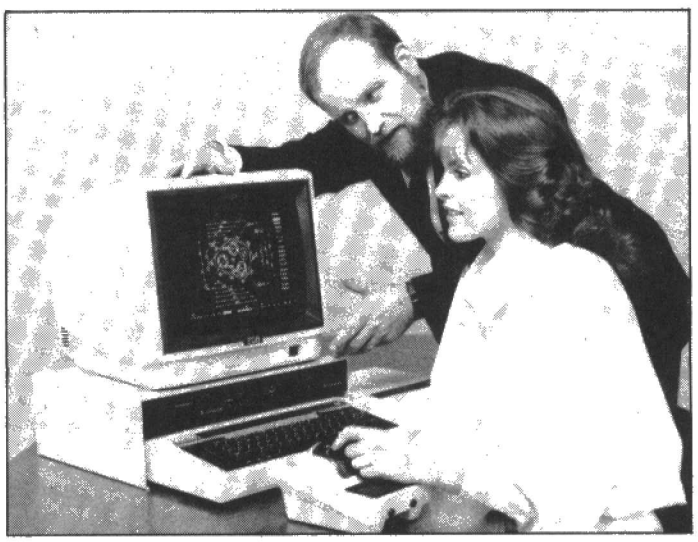
The first issue of *Computing Age*, on sale on September 13th will assess the Amiga computer in some depth. More on TRIPOS/AmigaDOS then. In the meantime the success of Metacomco in producing such a high quality piece of software on time only goes to show that the UK can still show a world beating lead in the field of software production.

Bitstik CAD system gets plotter interface

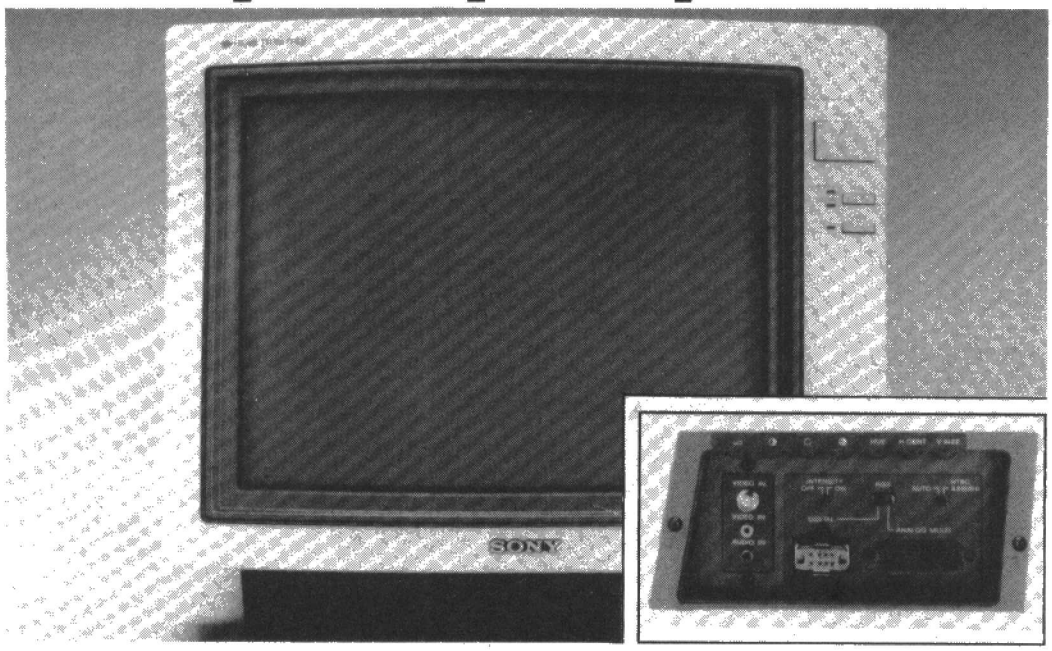
Many BBC users will be familiar with the Bitstik CAD system. Originally developed for the Apple computer, the system was launched for the BBC micro some time ago. A severe drawback with the mark one version of the Bitstick was that, while it allowed operators to create complex graphics on-screen, the software didn't support any interface to a plotter capable of producing hard copy. But the new system now takes care of this.

The Bitstick 2 costs £795 and owners of a level 1 system can upgrade to the new model for £450.

A full review of Bitstik 2 will appear in a future issue of *Computing Age*. In the meantime more details from *Robocom Limited, Clifton House, Clifton Terrace, London, N4 3TB. (Telephone 01 263 8585/272 8417).*



What price quality?



The answer if you're in the market for a computer display monitor is around £400. In the days of low cost colour monitors £400 may seem a high price to ask for a 14" design but this is what it will cost to get hold of Sony's new KX14CP1.

But the unit is packed with features that Sony hopes will justify the extra expense. In addition to adopting Sony's Triniton tube technology the monitor has a fine aperture grille which the company claims offers a superb clarity and reduces the effects of unwanted reflections.

Perhaps the most notable feature of the new monitor is the range of inputs. In addition to compatibility with all the composite video standards (PAL, NTSC and SECAM) the monitor has an eight pin RGB digital input and a 21 pin SCART RGB analogue connector. Add to this separate BNC and phono inputs for composite video and the Sony monitor should be able to cope with just about any signal format that a computer throws at it.

The monitor will be available from mid-August at your local Sony dealer.

In the picture

Video digitisers capable of accepting standard video signals as input and converting this analogue information into a stream of digital information in a form that can be placed into a computer's display RAM, have numerous applications. These range from the trivial such as the old favourite, picture printing on tee shirts to more serious applications in research and security.

For the average computer user such devices have been too expensive. Two recently released digitisers reverse this pattern by offering high specification designs at a reasonable price.

The first is for use with the BBC micro and is marketed by Watford Electronics. The package includes the digitiser unit, software in sideways ROM and a comprehensive user manual. The software includes a special print dump routine designed for production of fast, correctly proportioned picture prints.

Commodore 64 owners are catered for by the CRL digitiser. This again provides the interface and software to allow standard composite video images to be stored in the computer's RAM memory or written away to disk. Stored images may be processed and a printer driver allows the stored images to be printed out.

For more details of the BBC digitser which costs £89 plus VAT contact Watford Electronics at 250 High Street, Watford, WD1 2AN (telephone 0923 37774). The CBM 64 unit costs £149.95 and will be marketed by CBI of 7 Wings Yard, Carpenter's Road, London, E15 2HD (telephone 01 533 2918).

Eye in the sky – ear on the ground

There is increasing interest in linking home computers to satellite ground stations designed to receive the transmissions of the ever increasing number of satellite orbiting the earth.

MM Microwave Limited of York has recently announced the ASTRID satellite receiving system designed to decode and display data transmitted by the UOSAT satellites. The received data is recorded on a cassette recorder – the data may subsequently be displayed on a number of home computers.

Information transmitted by UOSAT includes experimental CCD TV camera signals, orbit information, satellite status and synthesised speech telemetry.

Astrid is priced at £144 and is supplied with comprehensive information regarding the reception of satellite signals.

For more details contact MM Microwave, Satellite Group, Thornton Road Industrial Estate, PICKERING, North Yorkshire. (Telephone 0751 75455).

Low cost disk drives

At £66 including VAT, RCS Computer Services' claim to be offering the lowest cost disk drive system for the BBC micro.

The unit is based on an Olivetti 5.25" drive providing a 100K, 40 track system. The drive is supplied with a utility disk, manual and all the cables required to connect it to a BBC micro fitted with Acorn's DFS.

Further details from RCS Computer Services (Leeway Data Products), Enterprise House, Central Way, North Feltham Trading Estate, Feltham, Middlesex, TW14 0RX. (Telephone 01-844 2044).

Increased penalties for pirates

The copyright (Computer software) Amendment Act received its Royal Assent in mid-July. The amendment brings computer software within the scope of the 1956 Copyright Act. The Law now provides for very stiff penalties for those convicted of offences concerned with the unauthorised duplication and distribution of computer software.

The maximum penalty for the manufacture, distribution or importing of pirated software is now an unlimited fine, or up to two years imprisonment, or both. Those selling, or even in possession of such counterfeit goods can be fined a maximum of £2000 for each offence.

The changes in the amendment up the stakes in the pirate game and should go some way to reducing the number of illegal copies of programs that are in circulation today.

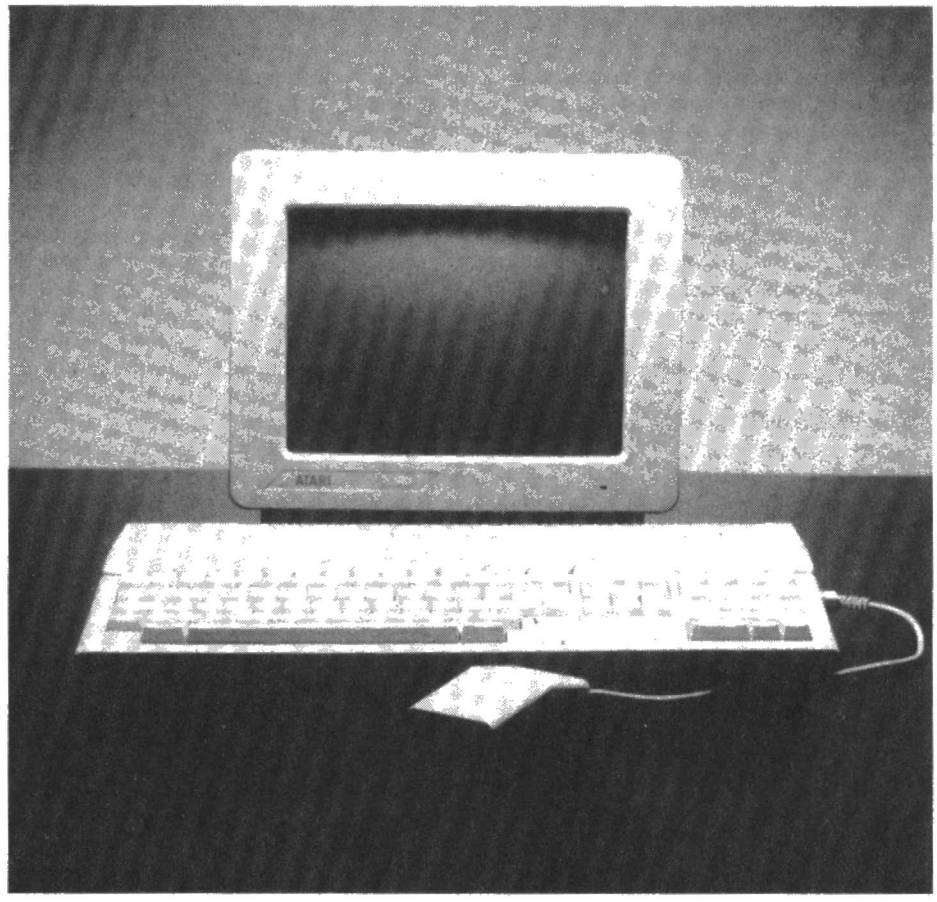
In late July Atari made a 520ST development system available to *E&CM* for a period of two weeks. The development system differs from those that will be available to the public only in the software supplied.

Development systems provide the Digital Research C compiler, the GEM Toolkit and usual CP/M-68K development tools such as an assembler, a linker and a symbolic debugger (the latter being a version of our old friend SID).

Naturally we would have preferred to have been supplied with the suite of applications software to be bundled with the final version of the system, but these were not available for review. Most were said to be only a couple of weeks away from completion, but with the exception of DR Logo and a brief glance at GEM Paint, we can't comment on the performance of these packages.

Accepting that any assessment of the 520 would be hampered by the fact that we couldn't evaluate the applications software we though it would be useful to see how the system performed in respects other than mere graphics ability.

With the resources made available to us in the form of the development system, the obvious approach was to write a few C programs – the BASIC wasn't ready either – to check the speed of the machine and to see how versatile its tools are. At this point Adam Denning takes up the story . . .



The Atari ST: a few problems still remain

GEM STILL FLAWED ON ATARI ST

Atari GEM is not ready yet. That's the conclusion Adam Denning came to after struggling with the ST development kit. He found that trying to run a few simple banchmarks in C was too much for the prototype computer.

We were severely hindered by a fundamental part of the system – TOS. This is the ST's own operating system, but is in effect merely a souped-up CP/M-68K, complete with FCBs, DMAs and archaic directory structures.

The problem was that it kept on crashing. Okay, this fact is well documented, and it has even been admitted that these early STs are best considered as *very* early! The second problem was that only a 'quick and dirty' editor was available. Now, CP/M being what it is, we could at least have expected to see a version of the infamous ED, couldn't we? Some companies developing software for the machine told us they had resorted to typing in the HEX bytes directly, using SID.

The programming editor supplied proved fairly useful. Except for another problem. Invoking the editor directly from the GEM Desktop started the program, but there was no cursor. This made the task of editing somewhat difficult, as the only way to tell where you were in a file was by pressing BACKSPACE until something deleted. Not entirely satisfactory. This problem was solved, in time, by reading through a mountain of software documentation. GEM, it seems, lies at the root of the problem. Unless it is told otherwise, all applications (ie programs) are considered as specifically GEM applications, which means they don't need a cursor.

All this could be alleviated, and a cursor gained, if the time was taken to install the program as a 'DOS – takes parameters' application. This involves selecting the application's icon, entering the 'Options' drop-down menu, selecting 'Install Appli-

cation' and finally clicking on the 'DOS takes parameters' box. Once that process has been completed, the installed application will behave much more sensibly. Whenever the program is invoked, it allows you to type a typical CP/M or MS-DOS command line, so you can actually give a name to the thing you are creating, and it also produces a cursor. But . . . hang on a minute! Wasn't GEM supposed to remove the need for complicated lines? Oh well.

"TOS . . . merely a souped up version of CP/M 68K . . ."

Worse than that, *every* application which needs to be accessed in this way must be installed, involving all those mouse presses per program, and if you want the installations to be permanent, you then have to save the Desktop on disk. If you go too far, things can get out of hand. On the PC version of GEM, I decided to install all the development tools. This has the advantage of giving you pretty icons, and more usefully, allows you to select a product of one of these applications, which will in turn invoke the application, thus obviating the need for the aforementioned command lines. So, I installed DEBUG.COM as an application, and said that it takes .COM files as its 'document type'. Instantly, all COM files became DEBUG documents, which meant that I couldn't run such fundamental things as COMMAND.COM as it was no longer considered an application. As I didn't realise this until after I had saved the Desktop, I was stuck with it. I still haven't discovered how to un-install an application!

Back to the ST. Having written a few valid-looking C programs using the screen editor, the next task was to transfer them across to the compiler disk for compilation. Now, our Atari has only one disk drive, but the operating system is sensible enough to know this. When it is asked to copy a file from disk A to disk B, it prompts at sensible moments for the operator to switch the disks in the drive. Every so often, it worked.

Atari still intends to launch the 520ST at the PCW show in September. The company's ability to fulfil this promise depends not on the hardware but on Digital Research's ability to complete its implementation of the GEM TOS and GEM application packages. We found nothing wrong with the Atari hardware during our tests, and came to the conclusion that the faults were caused by software birth pains – ie the TOS's crashing antics and the lack of a decent editor. It is to Atari's credit that it has not followed the example of one British manufacturer by launching a bug ridden machine on an unsuspecting public. Benchtests and software reviews will be published in *Computing Age* as soon as we receive the full production versions.

Most of the time it got as far as the last disk-swap request and then bombed out. Finally, with much perseverance, we got one of C files across. The next trick was compilation.

The C compiler, in common with all Digital Research C compilers, is comprised of a number of overlays. One for the pre-processor, one for the parser and one for the code generator. Normal stuff, it seemed. The MS-DOS DR C compiler is kind enough to invoke each in turn. The TOS compiler is not. And each section has a very tedious syntax which, even with correctly installed applications, means complicated command lines. We did discover a batch file, C.BAT, which seemed to be equipped to handle the whole lot. The question is, how do you run a batch file from TOS? CP/M programmers will no doubt be familiar with the SUBMIT program, and MS-DOS users will know that the MS-DOS command line interpreter, COMMAND.COM, is capable of executing .BAT files as automatically as it does .EXE and .COM files. We could find no program called SUBMIT.PRG on any of the TOS disks, and the command line interpreter, COMMAND.PRG, seemed unable to cope.

Eventually, we discovered a file called BATCH.PRG, which turned out to be just what we wanted. It had to be installed, of course, although it could have been run, tortuous command lines and all, from COMMAND.PRG. Anyway, the compilation finally proceeded, Compilation got more and more complicated. Each section of the compiler seemed to produce more than one output file, various permutations of which were needed by the subsequent sections. Throughout the process, files were being deleted automatically by the batch file.

The output of this compiler is not a relocatable object file, ready to be linked, as it is with other DR C compilers, but is instead an assembler source file which needs to be fed to the assembler. This too was done automatically by the batch file. The final result is a relocatable object file called <filename>.O, which can be linked. But more of that later.

As any C programmer knows, the general way to access a file in a C program is by using the fopen() function to return a pointer to type FILE, which uniquely identifies the open stream. Our problem was, the file which normally contains the definition of just what a FILE type is, STDIO.H, was nowhere to be seen. Perhaps it wasn't needed? No, that didn't work either. Eventually, we gave up trying to open files and reverted to the easier things such as

```
main ()
{
print f("\nHello world!\n");
}
and
main ()
{
   int i;
   for (i=0;i<32767;i++) printf("%x",i);
}
```

'This is the stuff reviews are made of', we thought!

Once the object files had been created, the linkage process needed to be started. This looked as though it may be almost as simple as compilation.

With two different linkers to choose from, we again searched for a batch file which could do the work for us. Finding this, it became apparent that we needed to copy our newly created object file to the linker disk so that we could generate, ultimately, the executable .PRG file. Although we were by now well aware of the hassles involved in disk to disk copies, the process had to be undertaken. Unfortunately, the linker disk turned out to have insufficient room for the object file (which was a surprising 9K long!), so we decided to go out and buy a few disks to create some working copies. We should have done this in the

ICONS AND WIMPS

GEM: THIS

Adam Denning explains how GEM should make life easier for both programmers and users.

Icons and WIMPs are this year's model. It all started with Apple's Lisa and Macintosh computers, but with Microsoft and Digital Research immediately taking up the cudgels for the PC market, things have really taken off.

So far, only Digital Research's product, GEM, has appeared; Microsoft's Windows is one of those products which has been coming 'real soon now' for months. GEM is available for the PC and most of its compatibles, and is standard on the Atari ST range.

Herein lies the key, as GEM is much more than just another WIMPs system. GEM stands for 'Graphics Environment Manager' and has the wonderful virtue of having the same programmer interface on each and every implementation. This means that if one were to write a program for GEM on the PC, it could be transferred to the ST and would run in an identical way.

Well, not quite. The condition is that the program is written in the C language. On the PC, it is easiest to use the Lattice C compiler, although Digital Research's own C compiler can be used. On the ST, the very same source file would be compiled on the C compiler which is supplied with the ST development system. Apart from demonstrating the portability of the C language and the intelligent way in which GEM has been written, it has the potential

"... most users will not need to use the C compiler ... The OS itself though is unsatisfactory ..."

first place, of course, but you know how it is when you've got a new toy!

This is when the final blow struck. After spending £37 on ten disks, we came back to the office to begin the process of copying requisite files from master to backup. Except ... the Atari refused to boot under any circumstances, and despite numerous attempts with three system disks, we haven't got the machine to boot since. After this, you begin to respect tacky dongles and microdrives.

The upshot of all this is that we are unable to bring you any benchmarks or approximate timings, except to say that the GEM Desktop itself far outperforms its 8088-based PC equivalent.

The experiences with the review system show that the 520ST is still some way from being ready for public consumption. While the majority of users will have neither the need nor the programming skill to make use of the C compiler, the operating system itself is still unsatisfactory. Atari is confident that by the time of the PCW show, the operating system and the bundled software will be ready for public launch. In the meantime we have been assured that once finalised versions of the software are available we will have a chance to reassess the performance of the 520 system.

All in all, I don't think that our experiences reflect upon Atari so much as Digital Research. GEM is that company's product, as is most of the other software provided with the ST. GEM provides a beautifully friendly interface between the computer and the *user*, but the interface between the GEM Toolkit and the *programmer* is truly horrific.

Another month's work by DR on the documentation, of what is certainly a radical and revolutionary product would have made the task of the programmer far easier.

The ST itself remains impressive but it does have the powerful threat of this month's wonder machine, the Amiga. *That* beast is certainly a better machine, and doesn't cost all that much more. The gold, it seems, is yet to glisten.

YEAR'S MODEL

to make a lot of software houses very rich as the same program can be sold for two entirely different machines with no extra development time.

The only additional expense is that you must have the GEM Programmer's Toolkit for each computer, as this contains the small section of assembler which interfaces with the VDI and the AES. The Toolkit costs around £500, which is peanuts to most software houses developing PC software.

The VDI is essentially the GEM device driver and the AES is the Applications Environment Services manager. They both have exactly the same interface, so once you have got to grips with the GEM system on one machine, you'll understand it on all the others.

The Toolkit also comes with the C 'bindings' for each GEM routine. These bindings are the calls and assignments needed for each operation, and they are the same for each GEM implementation. The technique would be to develop your programs, compile the C sections along with the relevant GEM interface routines, assemble the tiny sections of 8086 or 68000 assembly language, and then link them all together as an .EXE or .CMD file.

The number of routines available in the Toolkit is immense, and once you have got used to the graphics co-ordinate system you can do literally anything. The VDI is best seen as the successor to Digital Research's earlier graphics system, GSX. It allows programmers to treat each output device in exactly the same way, so the representation of a picture on the screen will be exactly the same on a printer or a plotter. This has been used in the applications programs, such as GEM DRAW, which allows you to build up pictures on the screen and then get camera-ready artwork out of your plotter.

The VDI consists of two major components – the device independent part, called GDOS ('Graphics Device Operating System') and the device-dependent device drivers for each output device. These are generally supplied with the GEM implementation. The IBM PC GEM, for example, comes with device drivers for Epson and IBM printers, the Hewlett-Packard plotter and numerous different display monitors and graphics card options.

A simple graphics program which used the VDI would be very trivial to write. A call (an INT on the PC and a TRAP on the 68000) to the VDI opens the 'workstation', and a further call could be used to draw an arbitrarily complicated figure. At the end of the program, the workstation needs to be closed. The tedious part of the code is filling each of the VDI control arrays with the parameters and co-ordinates, but apart from that there really isn't much to it.

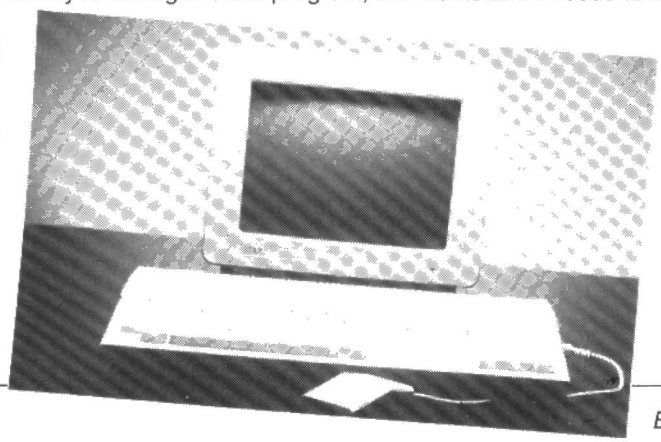
The AES contains the 'higher level' interface routines, such as the screen manager, the AES shell, the menu subroutines and the mouse routines. There is no requirement for an applications program to ever call or use any part of the AES, but the integration between the program and the GEM Desktop is far greater and more uniform if it does.

The Desktop is the standard application supplied with the VDI, but as it is the program which effectively implements the Mac-like interface on the PC, it is really rather more fundamental than 'just another application'. Most users of the GEM system will run all other programs from within the desktop, so that upon termination of the program, the machine re-enters the desktop rather than coming up with the familiar A>_ prompt.

All the physical operations such as changing a window's size, dragging it to another position, clicking on an icon and selecting it, or dragging an icon along the screen are handled by the subroutines in the AES library, but it is up to the application program to decide if the sizes or positions are actually valid and therefore accepted. For example, if you drag a file from a disk directory to a blank area of the workspace, it is the desktop itself rather than the AES or the VDI which objects to the action.

That GEM is here before the potentially better Microsoft Windows, and that the interface between programmer and device driver is so sensibly worked out and so easy to use must make the system a winner. If (as Digital Research must hope with a fervour equal to Atari's) the ST takes off, Microsoft and maybe even IBM itself will need to think again. And again.

GEM is much more than just another WIMPS system.



IMPORTANT ANNOUNCEMENT

COMPUTING AGE

First Issue on sale September 13th

From September 13th ***Electronics & Computing*** will become ***Computing Age.*** We thought readers of *E&CM* would like a sneak preview of the new magazine – we've therefore taken the wraps off the front cover of Issue One. We hope you like the look of ***Computing Age*** – remember the new title will incorporate all the popular elements of *E&CM* plus many new features of interest to the computer enthusiast.

In the First Issue . . .

Making the leap to 16-bits
A comparison of four leading low-cost 16-bit computers. That's the Apricot F1e, Sanyo MDC550, Sinclair QL and Atari 520ST.

Log on to Gold
A practical guide to electronic mail: how to get started; making the best use; and of course, how much does it really cost?

BBC EPROM blower
This project was originally published in *E&CM* two years ago. It proved so popular we've decided to run it again and make a number of improvements.

Compact disk storage
Or how to put all 24 volumes of the Encyclopædia Britannica on a single disk and still have room to spare.

Simple C
An introduction to the key language of modern computer systems.

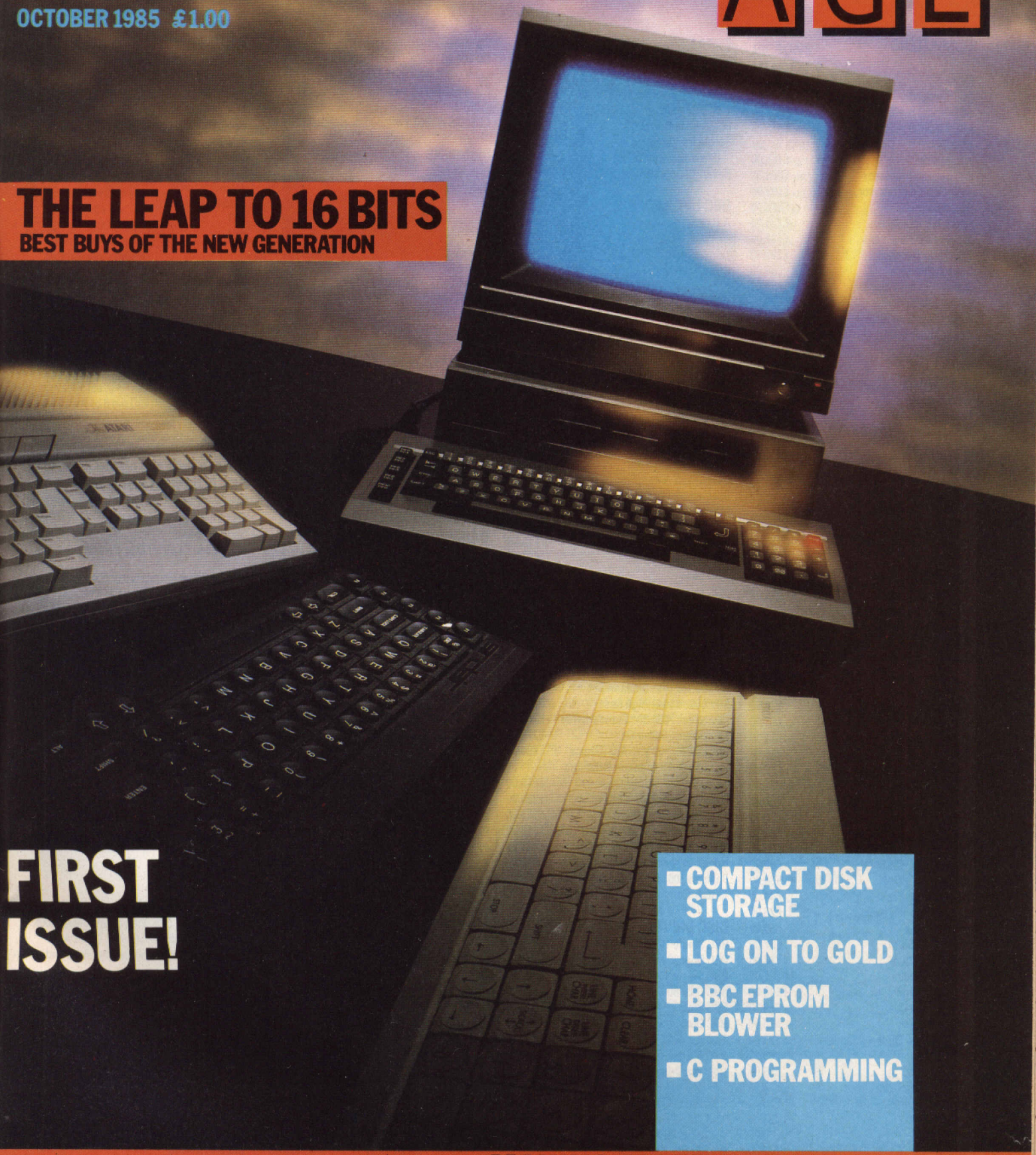
Plus: Using operating systems; The Commodore Amiga; Packet switching; Epson LX80 review; software for the BBC, QL and Amstrad, and more news, reviews, and features.

PLUS free 32 page guide to the latest in computer hardware.

Place an order with your newsagent now – demand for the first issue of ***Computing Age*** is bound to be great.

COMPUTING AGE

OCTOBER 1985 £1.00



THE LEAP TO 16 BITS
BEST BUYS OF THE NEW GENERATION

FIRST ISSUE!

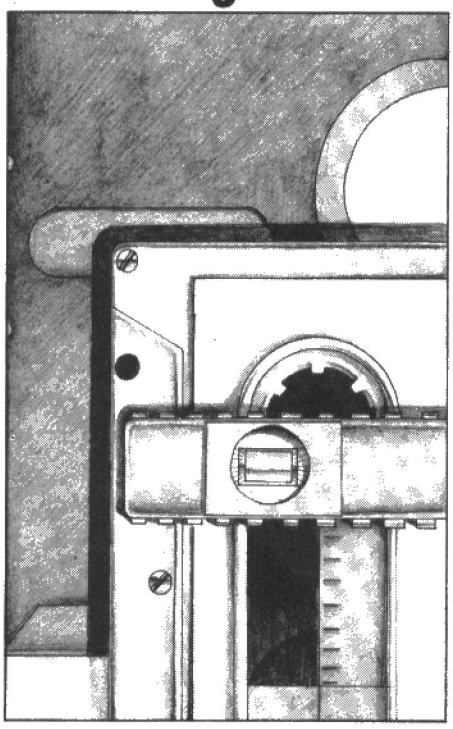
- COMPACT DISK STORAGE
- LOG ON TO GOLD
- BBC EPROM BLOWER
- C PROGRAMMING

MAKING MORE OF YOUR COMPUTER

AN EMAP PUBLICATION USA $2.95 GERMANY D6.00 SINGAPORE S$4.95

Drive control

Mike James continues his series on mass storage techniques with an in depth look at disk drive control and interfacing



How to add a new disk drive, daisy chaining, hard and soft sectoring, recording methods, formatting, and even the difference between double and single density disks are all examined in the latest installment of Mike James' series on mass storage media.

Last month's article looked at disk drive mechanics and maintenance but this is only the first part of the story. Once the read/write head is positioned over the track the question arises of what to record and how to read it back. The answer is found by following the signal processing chain that leads from the read/write head, through the disk controller electronics and into the computer. Although in principle it is the computer that decides how and what is recorded on a disk, disk controllers are becoming increasingly sophisticated and more and more involved in determining disk formats.

The control electronics used to operate a floppy disk are divided betwen the drive and computer. There is usually at least one PCB included within a disk drive to control its motors and to begin the process of interfacing with the computer; these are fairly primitive functions and the computer usually has a further chunk of circuitry dedicated to looking after the disk drive – the disk controller. The connection between the drive's electronics and the computer's disk controller are, for the computer industry, remarkably standard. Both 5.25" and 3.5" drives use a 34 pin PCB connector with roughly the same pin assignments, and most 8" disks use a 50 pin PCB connector. In other words 5.25" and 3.5" drives are generally plug compatible and you only have to watch out for a few 'difficult' brands of 8" disk drive.

The easiest way to appreciate what the disk drive's electronics do is to examine the purpose of each of the active connections between it and the computer's disk controller. Rather than deal with the connections in order of pin number it is logical to treat them according to what they do. But for reference purposes **Table 1** gives their functions in order of pin number for 5.25, 3.5 and 8" drives.

The READ DATA line simply passes a stream of pulses to the computer. These are derived by amplifying and processing the read head's output. That is, the pulses correspond to the pattern of magnetisation on the disk as it passes under the read head – wherever it might be on the disk. Notice that the read data line presents a stream of pulses, not data, to the computer. It is the responsibility of the disk controller to convert the pulses to useful data.

The INDEX/SECTOR pulse line is the amplified and 'squared up' output of the index sensor (see last month's article *Remedies for a sick disk drive*). Each time the disk's index hole passes under the index sensor a short pulse (approx 40ms for a 5¼" disk) is output on this line.

The index/sector pulse line is sometimes used by disk controllers to detect the presence of a disk in drives that do not have a READY line (see later). Index/sector pulses indicate that a disk is in place and is spinning.

The TRACK 00 line is the output from the track zero switch. It is used by the disk controller to tell when the head is

TABLE 1

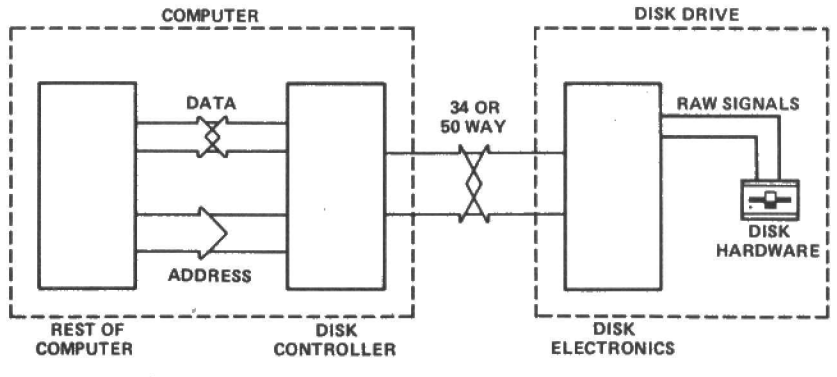
Pin number 5.25" (8")	Function
4 (18)	Head load
6 (32)	DS4 drive select 4
8 (20)	Index – index/sector pulses
10 (26)	DS1 – drive select 1
12 (28)	DS2 – drive select 2
14 (30)	DS3 – drive select 3
16	Motor on
18 (34)	Step Direction
20 (36)	Step pulse
22 (38)	Write data
24 (40)	Write gate
26 (42)	Track zero pulse
28 (44)	Write protect
30 (46	Read data
32 (14)	Side select
34 (22)	Ready

Note: all odd numbered pins are ground.

The standard control lines to the computer are:

READ DATA
INDEX/SECTOR PULSE
TRACK 00
WRITE PROTECT

Figure 1. The division of the disk drive electronics.



positioned over track zero.

The WRITE PROTECT line is the output from the write protect switch. The disk controller tests the status of this line to discover if it is permissible to write on a disk.

The input lines to the disk drives are more numerous but just as simple:

WRITE GATE
WRITE DATA
MOTOR ON
DIRECTION SELECT
STEP

The WRITE GATE sets the drive either to read or to write a disk. If the write gate line is set low then the drive's write electronics are enabled.

The WRITE DATA line is used by the disk controller to send a stream of pulses to the drive's write amplifiers. Of course this stream of pulses only reaches the head if the write gate line has first been set low. As in the case of the read data, the stream of pulses has no meaning to the disk drive and the way that it is derived from the data is entirely a matter for the disk controller.

The MOTOR ON line does exactly what its name suggests. A low on this line turns the drive's motor on and starts the diskette spinning. Some drives take no notice of this line as their motors are kept permanently running. Some drives also load the read/write head as the the motor is started (but even so it takes about a second for the disk to come up to speed).

The DIRECTION SELECT line determines the direction that the head will move in response to the next pulse on the step line. If it is low the head is moved towards the centre of the disk.

The STEP line causes the head to move by one track in the direction determined by the direction select line. A pulse on the step line then causes the head to move.

In a system using only a single drive the interface lines described above are sufficient, but if there is more than one drive in use then the disk controller must have some way of selecting one of them. The DRIVE SELECT lines DS1 to DS4 are provided to do just this. When more than one drive is used it is normal to connect all of the interface lines to all of the drives; this is called 'daisy chaining'.

The drive select lines DS1 to DS4 are also connected to each drive but only one drive is configured to respond to each line. A drive is selected and will respond to the disk controller's commands if its drive select line is low; if it is high then all signals on the input lines are ignored and no signals are entered into the output line.

There are a number of other lines that are occasionally found: HEAD LOAD, SIDE SELECT and READY lines are very common.

Recording Methods – FM and MFM

The method of representing data as a stream of bits to be recorded on a floppy disk is up to the disk controller. All that the drive does is faithfully record the pulses that are sent to it and reproduce them on demand. Due to the size of the head gap (see *Making a mark E&CM* July '85) there is a lower limit to the size of pulse that can be recorded reliably using a given drive. The size of this pulse obviously determines how much data can be recorded on a disk, and disk drives are made in two types – single density and double density.

The only difference between a single and double density drive is the size of the head gap. You can use a double density drive in a single density application but, obviously, not vice versa.

Fortunately, apart from one or two exceptions such as Apple, most computers use either the FM (Frequency Modulation) or the MFM (Modified Frequency Modulation) method of recording data. All that is needed is a representation of 0 and 1 that can be easily decoded. The FM method is generally used for single density recording and it works by recording the data and a stream of clock pulses (illustrated in **Figure 2a**). The clock and data pulses are easily recovered from the data stream using either a simple mono-stable or a phase locked loop. The MFM method is a little more complicated in that no clock pulses are recorded unless a pair of zeros occurs together (see **Figure 2b**).

'Apart from one or two exceptions such as Apple, most computers use the same recording method'.

Formatting and soft sectoring

Now that we have a method of recording information on a disk, the next question is how to organise the storage of data. Each track could be treated as a unit of data storage and one complete track read and written to at a time, but this involves a large

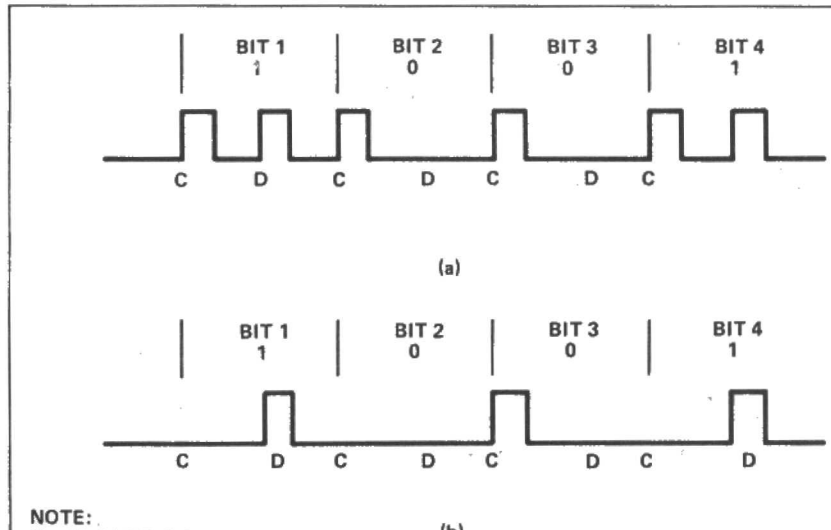


Figure 2(a) FM coding; (b) MFM coding.

amount of data (approximately 2K). A better method is to divide each track into a number of smaller units called sectors. The number of sectors into which a track is divided varies between systems. Typical sector sizes are 128, 256 or 512 bytes. Using a sectored disk any data can be found by giving two pieces of information – its track number and its sector number. (The track and sector number are often quoted together and referred to as a 'disk address'.)

It is easy to see how the drive can find any given track: starting from a known track – track zero say – it simply steps in the required number of tracks. But how does it find a given sector?

There are two methods of indicating which sector is about to pass under the read head: hard sectoring and soft sectoring. A hard sector disk uses 'sector holes' to mark the start of each sector in the same way that the index hole marks the start of each track. Finding a given sector is simply a matter of counting the number of sector holes that have passed under the index sensor since the last index pulse. For various reasons hard sectored disks are not used very much these days and most systems use soft sectoring.

A soft sectored disk has information recorded on each track that allows the drive to read the number of the data sector about to pass under the read head. Each sector is in fact made up of two portions, an ID field and a data field. The ID field contains unchanging information about the nature of the sector and is not re-written during normal disk operations. To create the pattern of ID and data fields on a diskette it has to be formatted, and the format program is generally the only software that ever writes ID fields. You can think of ID fields as a sort of magnetic equivalent of the sector holes used by hard sectored disks, but ID fields are more versatile because they can be read to discover the sector number of the data about to pass under the read head.

There are a number of different ways to format a disk but nearly every system either uses or can use a format that is close to the IBM 3740 standard. You can see the exact construction of both the ID and data fields that make up a sector in **Figure 3.** The track number is also recorded in each ID field to enable the head's current position to be verified. The length of the data portion of the sector is also recorded as a simple code: 00 = 128 bytes; 01 = 256 bytes; 02 = 512 bytes; 03 = 1024 bytes.

For synchronisation purposes there are a number of special data patterns or address marks that are recorded on the disk. These address marks correspond to sequences of pulses that can't normally occur because they lack various clock pulses; in that way the disk hardware can identify an address mark without any doubt. Different address marks are used to signify the start of the ID field (the ID address mark) and the data field (the data address mark). At the end of each of the ID and data fields are two Cyclic Redundancy Check (CRC) bytes, used to make sure that data has been read without error. CRC bytes are computed when any field is written, and recomputed when any field is read. If the recomputed values don't match the recorded values then the data has changed since it was written and a read error should be reported.

As well as address marks signifying the start of each field there must also be gaps between the fields and between the sectors to allow for timing inaccuracies. There must also be a gap following the index hole at the start of a track and a gap after the last sector before the index hole. Thus there are four types of gap and each has a unique format to help with the detection of the address marks. The complete format of a track can be seen in **Figure 4.** The standard IBM 5.25" format gaps are made up as follows:

1	**Post index**	22 bytes made up of 16 FFH followed by 6 00H
2	**ID**	17 bytes made up of 11 FFH followed by 6 00H
3	**Data**	32 bytes made up of 26 FFH followed by 6 00H
4	**pre-index**	274 bytes made up of FFH

When you consider all of the information that is recorded on a disk other than the data in the data field you can appreciate that the unformatted capacity of a disk is much more than its formatted capacity.

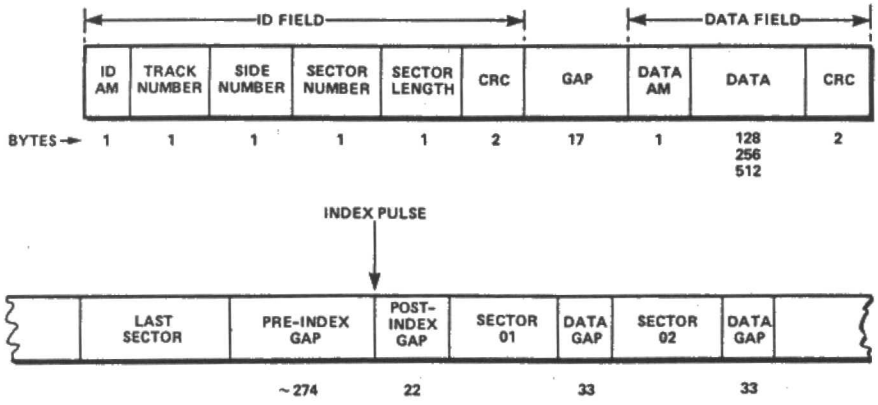


Figure 3 (top) IBM sector format.
Figure 4 (bottom) IBM track format.

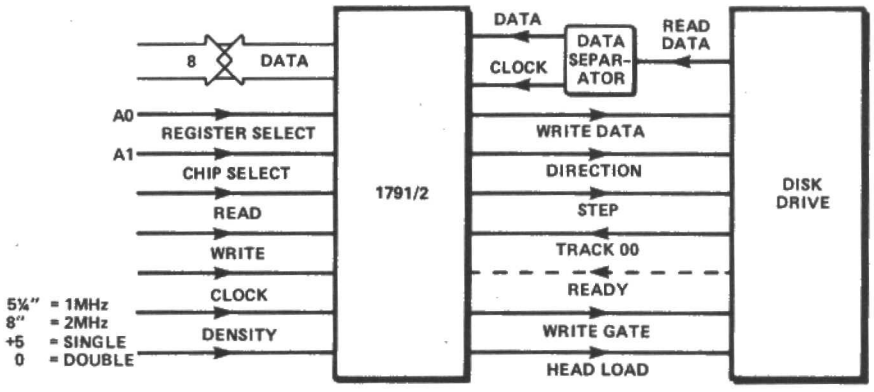
The disk controller

Now that the details of the disk's own electronics and the intricacies of soft sectoring have been explained you should be able to see what is left for the disk controller to do. These days most disk controllers are designed around a single chip – either the 8271 or one of the Western Digital 179X controllers. If you need to know how one of these chips works then there is no substitute for a data sheet, but a short explanation of the 1791/2 follows.

A typical circuit for a 1791/2 disk controller can be seen in **Figure 5.** From the software point of view the controller looks like a number of registers – command, status, track, sector and data to be precise. The command register and the status register are used by the computer to give the disk controller its instructions and to find out how they are progressing. The track and sector register are used to hold the disk address of the sector that an instruction refers to and the data register is used to send or receive data from the disk drive.

Given a disk that is already formatted, the operations needed to write some data to a sector are straightforward. If the track register contains the number of the track that the head is currently assumed to be

Figure 5. Simplified 1791/2 disk controller interface.



over, then the number of the track required is stored in the data register and a SEEK command code is stored in the command register. A SEEK command causes the disk controller to set the stepping direction and to output stepping pulses until the track register contains the same number as the data register. If at any point during disk operation the track register is found not to contain the track number then an error is reported and usually the software will issue a RESTORE command to step the head out until the track zero switch is activated. This moves the head to track zero so that the track register can be cleared (sometimes referred to as calibration).

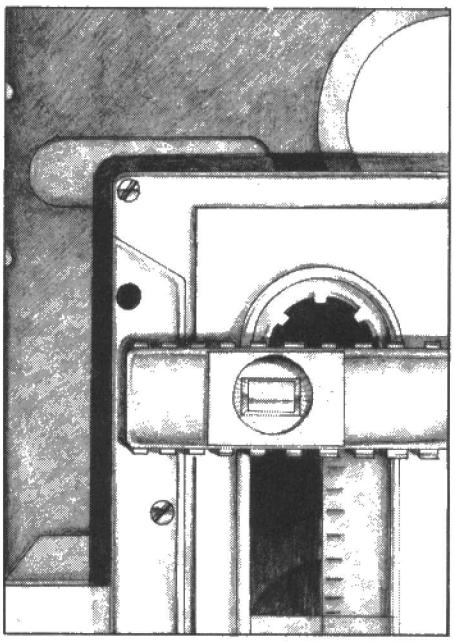
Once the head is positioned over the desired track the sector register is loaded with the desired sector and a READ SECTOR command is issued. This causes the disk controller to read the ID fields as they pass under the head until one with the correct track and sector number is found. Then the data in the corresponding data field is read in. If an ID field with the right track and sector is not found within one revolution an error is reported and the disk software usually issues a RESTORE command and tries a few times before giving up and reporting a disk error to the user. Writing a sector follows the same stages of head positioning and of reading ID fields to find the correct sector only; instead of reading the data in, the new data is written out to the data field.

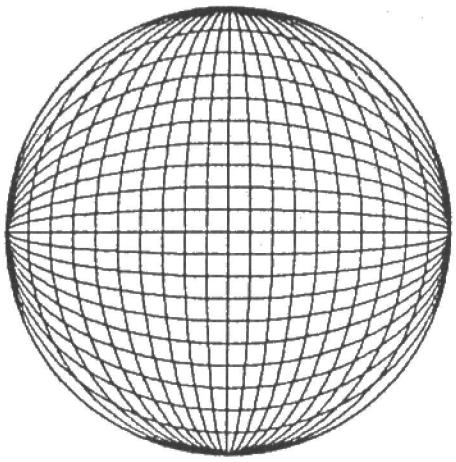
The only remaining problem is how a disk ever gets formatted in the first place. The ID fields cannot be written using a WRITE SECTOR command because this only changes the data in the data field. The answer is that the 1791/2 and most other controller chips include a READ TRACK and a WRITE TRACK command. Both of these commands ignore the sector structure of a track and simply take the index pulse as a marker fro the start of a track which is then either read or written in its entirety. To format a disk the formatter simply writes the track pattern described earlier onto each track in turn using the WRITE TRACK command.

'there are a number of ways to format a disk but nearly every system uses the IBM 3740 standard'.

Working upwards

We have come a long way since the description of the basic disk mechanism in last month's article but there is still some way to go before the level of disk and program files is reached. So far, all that the disk drive and disk controller can do for us is store and recall chunks of data in the form of sectors. The final stage in making the floppy disk useful is converting this collection of sectors into named files. This is where hardware and electronics gives way to software – the DOS or Disk Operating System. And the dirt on DOS comes next month.





Proceed to plot

This range of simple plotter proceedures for the BBC Micro has been written by Paul Beverley to work with the Tandy X-Y plotter, but can be easily adapted to any other make.

As a result of its association with ACT, Tandy has off-loaded a lot of perfectly good A4 X-Y plotters. One of them came my way and most of the rest are still being sold off. I've since written a range of procedures on a BBC micro for use with this plotter, but most of them should work with any type of plotter by re-writing the procedure that turns the parameters into commands.

The basic function of the procedures shown in **Listing 1** is to draw pictures and diagrams made up of geometric shapes: rectangles, polygons, circles, ellipses, arcs, tangents, chords, radii and so on. After clearing the screen and initialising all the variables within the system, the rest of the program will consist of your own calls to the various procedures listed as lines 10010 onwards.

As an example of how simple the programs can be, I have included two which draw the flags of Canada and USA: **Listings 2 and 3;** and a program to draw a sphere with a 3-D effect: **Listing 4.**

Listing 5 is the source code for a machine code driver routine to enable you to use the serial interface rather than the Centronics parallel interface. The plotter has both types of interface, but it is much cheaper to make up a lead with a couple of DIN plugs and a bit of three-core mains cable than to buy a Centronics ribbon cable. You need the machine code driver because the implementation of RS232 on Tandy plotters and printers is not as versatile as it might be, with the result that they do not operate properly with non-Tandy computers.

The rest of this article is a description of the operation of various procedures to use in your own plotting routines.

PROCmove(x%,y%), PROCdraw(x%,y%) The purpose of these two procedures should be fairly obvious. But note that the values used go through a three stage scaling process using **Xscale, Yscale** and **scale.** Unless you change them all three scale factors are unity. Changing **Xscale** to 0.5 for example will shrink the whole picture to half its width. Using **scale=0.5** will shrink the whole picture to half linear size, ie one quarter in terms of area.

The absolute values of the co-ordinates that you can use are in the ranges: x% = 0 to 2700; y% = 0 to 1860. If you try to go further, the pen just lifts off the paper. The numbers specify the position in tenths of millimetres, ie 27cm x 18.6cm.

If you use **PROCbigsheet** you can use the larger area of 2980 x 2160 (29.8cm x 21.6cm), and then **PROCsmallsheet** switches back to A4 size.

If you want to use positive and negative co-ordinates you can move the origin to a new position on the sheet. Henceforth all procedures work from the new origin. This is done by using **PROCorigin(ox%,oy%).**

PROCdots(len%), PROCdodots. The lines which the plotter draws can either be dotted or full. When you switch to a dotted line, you can select the dot length with **len%.** The default is no dots.

PROCXaxis(length%,tic%), PROC Yaxis(length%,tic%) You can draw your axes for a graph automatically. Simply specify the total length of the axis and the space between marks. The starting point is the current position of the pen.

PROChome Just moves the arm off the paper, (bottom right) so that you can admire your masterpiece or change the sheet of paper. Function key 0 is programmed to call this procudre in immediate mode.

PROCchange(pencol$) If you want to stop the program and prompt for a pen change, this procedure prints up an instruction to change pens to the one specified by **pencol$.**

PROCprint(N$,size%,dirn%) Prints **N$,** starting at the current position of the pen, with characters of a given size **(size%),** and in one of four directions (0 to 3). The character size is scaled with the rest of the diagram.

The remaining procedures all require the setting of the global variables **X%** and **Y%.** These can be set using assignments **"X%=",** and **"Y%="** or by using, **PROCset(x%,y%), PROCshift(dx%,dy%).** The shift procedure increases or decreases the values of **X%** and **Y%** by the specified amount, eg **PROCshift(2000,–500)** changes the values of **X%** and **Y%** by 200 to the right and 500 down but does not move the pen itself. NB **PROCmove** and **PROCdraw** do not change the values of **X%** and **Y%.**

PROCbox(h%,v%,Ha%) Having set X% and Y%, this procedure draws a rectangle **h%** wide and **v%** high, with or without hatching. **Ha%=1** for horizontal hatching, **Ha%=2** for vertical hatching, **Ha%=4** for diagonal hatching sloping down to the right and **Ha%=8** for diagonal hatching sloping up to the right. Any combination of types of hatching can be made by adding up the values. Thus **Ha%=12** = (4+8) will give cross diagonal hatching and **Ha%=3** = (1+2) will give squared hatching. The hatch spacing (set initially at 20) can be changed with the global variable **hatch%.**

Figure 1. Drawing produced by Listing 2.

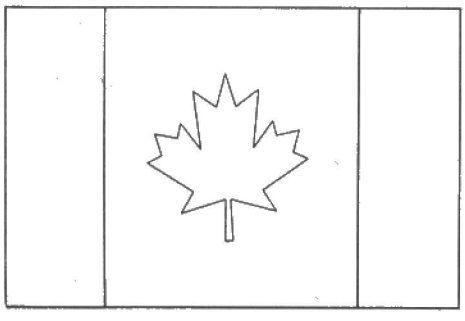
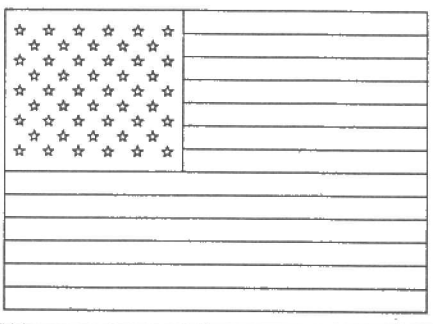


Figure 2. Drawing produced by Listing 3.



LISTING 1. Plotter procedures.

```
   10 MODE7
   20 PROCinitialise
10000 END
10010
10020 DEF PROCset(x%,y%)
10030 X%=x%:Y%=y%
10040 ENDPROC
10050
10060 DEF PROCmove(x%,y%)
10070 x%=FNX(x%)
10080 y%=FNY(y%)
10090 PROCsend("M"+STR$(x%)+","+STR$(y%))
10100 ENDPROC
10110
10120 DEF PROCdraw(x%,y%)
10130 x%=FNX(x%)
10140 y%=FNY(y%)
10150 PROCsend("D"+STR$(x%)+","+STR$(y%))
10160 ENDPROC
10170
10180 DEF PROCbigsheet
10190 PROCsend("F1")
10200 ENDPROC
10210
10220 DEF PROCsmallsheet
10230 PROCsend("F0")
10240 ENDPROC
10250
10260 DEF PROChome
10270 PROCsend("H")
10280 ENDPROC
10290
10300 DEF PROCnoeccen
10310 eccenX=1
10320 eccenY=1
10330 ENDPROC
10340
10350 DEF PROCorigin(ox%,oy%)
10360 ox%=FNX(ox%)
10370 oy%=FNY(oy%)
10380 PROCsend("I"+STR$(ox%)+","+STR$(oy%))
10390 ENDPROC
10400
10410 DEF PROCshift(dx%,dy%)
10420 X%=X%+dx%
10430 Y%=Y%+dy%
10440 ENDPROC
10450
10460 DEF PROCprint(N$,size%,dirn%)
10470 PROCsend("O"+STR$(dirn%))
10480 IF size% PROCsend("S"+STR$(INT(size%*scale)))
10490 PROCsend("P"+N$)
10500 ENDPROC
10510
10520 DEF PROCdots(len%)
10530 IF len% PROCsend("B"+STR$(INT(len%*scale)))
10540 PROCsend("L1")
10550 ENDPROC
10560
10570 DEF PROCnodots
10580 PROCsend("L0")
10590 ENDPROC
10600
10610 DEF PROCXaxis(length%,tic%)
10620 step%=length%/tic%*scale*Xscale
      "+STR$(step%))
10630 PROCsend("X1,"+STR$(INT(tic%*scale*Xscale))+",
10640 ENDPROC
10650
10660 DEF PROCYaxis(length%,tic%)
10670 step%=length%/tic%*scale*Yscale
10680 PROCsend("X0,"+STR$(INT(tic%*scale*Yscale))+
      ","+STR$(step%))
10690 ENDPROC
10700
10710 DEF PROCchange(pencol$)
10720 VDU 7,12
10730 PRINT"Please change to "¡pencol$
10740 PRINT''"Press space when ready."
10750 REPEAT UNTIL GET=32
10760 PRINT'
10770 ENDPROC
10780
10790 DEF PROCbox(h%,v%,Ha%)
10800 PROCmove(X%,Y%)
10810 PROCdraw(X%+h%,Y%)
10820 PROCdraw(X%+h%,Y%+v%)
10830 PROCdraw(X%,Y%+v%)
10840 PROCdraw(X%,Y%)
10850 IF (Ha% AND 1) PROChhatch
10860 IF (Ha% AND 2) PROCvhatch
10870 IF (Ha% AND 4) PROCdownhatch
10880 IF (Ha% AND 8) PROCuphatch
10890 ENDPROC
10900
10910 DEF FROCpoly(rp%,sides%,as)
10920 PROCpoint(rp%,as)
10930 PROCmove(A%,B%)
10940 da=360/sides%
10950 FOR s%=1TOsides%
10960   PROCpoint(rp%,as+s%*da)
10970   PROCdraw(A%,B%)
10980   NEXT
10990 ENDPROC
11000
11010 DEF PROCstar(ri%,ro%,points%,as)
11020 PROCpoint(ro%,as)
11030 PROCmove(A%,B%)
11040 da=360/points%
11050 FOR s%=1TOpoints%
11060   PROCpoint(ri%,as+(s%-0.5)*da)
11070   PROCdraw(A%,B%)
11080   PROCpoint(ro%,as+s%*da)
11090   PROCdraw(A%,B%)
11100   NEXT
11110 ENDPROC
11120
11130 DEF PROCchord(rch%,a1,a2)
11140 PROCpoint(rch%,a1)
11150 PROCmove(A%,B%)
11160 PROCpoint(rch%,a2)
11170 PROCdraw(A%,B%)
11180 ENDPROC
11190
11200 DEF PROCspoke(ra%,asp)
11210 PROCpoint(ra%,asp)
11220 PROCmove(A%,B%)
11230 PROCdraw(X%,Y%)
11240 ENDPROC
11250
11260 DEF PROCcircle(rc%)
11270 oldrot=angrot
11280 PROCarc(rc%,0,360)
11290 angrot=oldrot
11300 ENDPROC
11310
11320 DEF PROCellipse(maj%,min%,rotn)
11330 oldeccenX=eccenX
11340 oldeccenY=eccenY
11350 oldrot=angrot
11360 angrot=rotn
11370 eccenY=min%/maj%
11380 eccenX=1
11390 PROCarc(maj%/2,0,360)
11400 angrot=oldrot
11410 eccenX=oldeccenX
11420 eccenY=oldeccenY
11430 ENDPROC
11440
11450 DEF PROCarc(ra%,a1,a2)
11460 da=360/ra%*2/scale*arcscale
11470 as=a1:af=a2
11480 IF a1>a2 as=a2:af=a1
11490 PROCpoint(ra%,as)
11500 PROCmove(A%,B%)
11510 FOR a=as TO af+da STEP da
11520   PROCpoint(ra%,a)
11530   PROCdraw(A%,B%)
11540   NEXT
11550 ENDPROC
11560
11570 DEF PROCpoint(ra%,ap)
11580 A=ra%*COS(ap*dr)*eccenX
11590 B=ra%*SIN(ap*dr)*eccenY
11600 IF angrot PROCrotate
11610 A%=X%+A
11620 B%=Y%+B
11630 ENDPROC
11640
11650 DEF PROCrotate
11660 rr=SQR(A*A+B*B)
11670 IF A=0 angle=PI/2 ELSE angle=ABS(ATN(B/A))
11680 IF A<0 angle=PI-angle
11690 IF B<0 angle=-angle
11700 angle=angle+angrot*dr
11710 A=rr*COS(angle)
11720 B=rr*SIN(angle)
11730 ENDPROC
11740
11750 DEF FNX(xs%)
11760 =xs%*scale*Xscale
11770
11780 DEF FNY(ys%)
11790 =ys%*scale*Yscale
11800
11810 DEF PROChhatch
11820 FOR hh%=Y%+hatch% TO Y%+v%-2 STEP hatch%
11830   PROCmove(X%,hh%)
11840   PROCdraw(X%+h%,hh%)
11850   NEXT
11860 ENDPROC
11870
11880 DEF PROCvhatch
```

LISTING 1 (Continued)

```
11890 FOR vv%=X%+hatch% TO X%+h%-2 STEP hatch%
11900   PROCmove(vv%,Y%)
11910   PROCdraw(vv%,Y%+v%)
11920   NEXT
11930 ENDPROC
11940
11950 DEF PROCdownhatch
11960 xs%=X%:ys%=Y%:xf%=X%:yf%=Y%
11970 xb%=X%+h%:yb%=Y%+v%
11980 pcx=0:pcy=0
11990 REPEAT
12000   IF pcx ys%=ys%+hatch% ELSE xs%=xs%+hatch%
12010   IF xs%>xb% pcx=1:ys%=Y%+xs%-xb%:xs%=xb%
12020   IF pcy xf%=xf%+hatch% ELSE yf%=yf%+hatch%
12030   IF yf%>yb% pcy=1:xf%=X%+yf%-yb%:yf%=yb%
12040   PROCmove(xs%,ys%)
12050   IF ys%<yb% PROCdraw(xf%,yf%)
12060   UNTIL ys%>=yb%
12070 ENDPROC
12080
12090 DEF PROCuphatch
12100 xs%=X%+h%:ys%=Y%:xf%=X%+h%:yf%=Y%
12110 yb%=Y%+v%
12120 pcx=0:pcy=0
12130 REPEAT
12140   IF pcx ys%=ys%+hatch% ELSE xs%=xs%-hatch%
12150   IF xs%<X% pcx=1:ys%=Y%+X%-xs%:xs%=X%
12160   IF pcy xf%=xf%-hatch% ELSE yf%=yf%+hatch%
12170   IF yf%>yb% pcy=1:xf%=X%+h%-yf%+yb%:yf%=yb%
12180   PROCmove(xs%,ys%)
12190   IF ys%<yb% PROCdraw(xf%,yf%)
12200   UNTIL ys%>=yb%
12210 ENDPROC
12220
12230 DEF PROCsend(cmd$)
12240 VDU2
12250 PRINT cmd$
12260 VDU3
12270 ENDPROC
12280
12290 DEF PROCinitialise
12300 ONERROR PROCerror:END
12310 PROCsend("F0")
12320 PROCsend("S4")
12330 PROCsend("L0")
12340 PROCsend("Q0")
12350 scale=1
12360 Xscale=1
12370 Yscale=1
12380 hatch%=20
12390 dr=2*PI/360
12400 arcscale=1
12410 eccenX=1
12420 eccenY=1
12430 angrot=0
12440 *KEY0PROChome M
12450 *KEY1L.,9999 M
12460 ENDPROC
12470
12480 DEF PROCerror
12490 VDU3
12500 REPORT
12510 PRINT" at line ";ERL
12520 ENDPROC
```

All the circular and eliptical shapes use X%, Y% as their centre and are controlled by global variables: **arcscle, eccenX, eccenY** and **angrot.** All the curves have to be made up of a succession of short straight lines and **arcscale** sets the length of the lines of which the curves are made up. It does not set the absolute length of the lines but is related to the radius of the arc. You will find that **arcscle=3** is the most course setting tolerated.

The two eccentricity variables stretch all the acrs, circles etc in the two directions. (This is independent of the distortion caused by **Xscale** and **Yscale.**) If **eccenX** and **eccenY** are not unity or either **Xscale** or **Yscale** is changed, **PROCcircle** will produce an elipse. **PROCnoeccen** just switches off the eccentricity ie it makes **eccenX** and **eccenY** both equal to 1. All the circular functions are subject to an angular rotation set by the global variable **angrot.**

PROCchord(rch%,a1,a2), PROCspoke (ra%,asp), PROCarc(ra%,a1,a2) Chords and arcs are specified by their radius and the start and end angles. Spokes (or radii) are specified only by radius and single angle. (All absolute angles are specified in the clockwise direction, measuring from the positive X-axis and are quoted in degrees.) These can all be made eccentric and can also be rotated, although, without eccentricity, the rotation of arcs and chords would be pointless. They could be more easily rotated by choosing different values of **a1** and **a2.**

PROCcircle(rc%) simply draws a circle centre **X%, Y%** of radius **rc%.** Elipses can be made by using **PROCcircle** with eccentricity specified by **eccenX** and/or **eccenY** and with angular rotation given by **angrot,** but they can also be drawn by using a separate procedure, **PROCellipse(maj%, min%,rotn)** where the lengths of the major and minor axes are specified, as is the rotation of the major axis away from the X-axis. This procedure does not change the current value of **angrot.**

LISTING 2. Maple Leaf.

```
 10 MODE7
 20 PROCinitialise
 30 PROCbigsheet
 40 PROCset(200,200)
 50 PROCbox(2600,1600,0)
 60 PROCmove(850,200)
 70 PROCdraw(850,1800)
 80 PROCmove(2150,200)
 90 PROCdraw(2150,1800)
100 PROCset(1500,500)
110 PROCmove(X%,Y%)
120 REPEAT
130   READ X1%,Y1%
140   IF X1% OR Y1% PROCdraw(X%+X1%,Y%+Y1%)
150   UNTIL X1%=0 AND Y1%=0
160 RESTORE
170 PROCmove(X%,Y%)
180 REPEAT
190   READ X1%,Y1%
200   IF X1% OR Y1% PROCdraw(X%-X1%,Y%+Y1%)
210   UNTIL X1%=0 AND Y1%=0
220 DATA 25,0
230 DATA 15,250
240 DATA 290,170
250 DATA 210,300
260 DATA 490,480
270 DATA 400,520
280 DATA 440,650
290 DATA 300,620
300 DATA 280,710
310 DATA 160,550
320 DATA 200,900
330 DATA 60,800
340 DATA 0,1000
350 DATA 0,0
360 END
```

LISTING 3. Stars and Stripe.

```
 10 MODE7
 20 PROCinitialise
 30 hatch%=153
 40 width=1200
 50 scale=0.9
 60 PROCset(0,0)
 70 PROCbox(2850,6*hatch%,1)
 80 PROCset(width,6*hatch%)
 90 PROCbox(2850-width,7*hatch%,1)
100 PROCmove(1200,1989)
110 PROCdraw(0,1989)
120 PROCdraw(0,6*hatch%)
130 startX=100
140 PROCset(startX,1050)
150 W%=100
160 PROCmove(X%,Y%)
170 FOR row=1TO9
180   stars=5
190   IF row MOD2 stars=6 ELSE PROCshift(W%,0)
200   FOR star=1TOstars
210     PROCstar(14,40,5,90)
220     PROCshift(2*W%,0)
230     NEXT
240   PROCset(startX,Y%+W%)
250   NEXT
260 END
```

LISTING 4. Globe.

```
 10 MODE7
 20 PROCinitialise
 30 PROCset(1200,900)
 40 FORN=1TO2
 50   FOR eccenY=0 TO 1.01 STEP 0.1
 60     PROCcircle(400)
 70     NEXT
 80   angrot=90
 90   NEXT
100 END
```

PROCpoly(rp%,sides%,as) produces a regular polygon based on the radius of the circle that would join its points. It can have any number of sides and so if **sides%** is large, this could also be used to produce what is effectively a circle. The angle at which you start to draw the polygon is specified by **as.** Thus **PROCpoly(100,4,45)** would produce a square with sides of length 100 with its sides parallel to the axes, whilst **PROCpoly(100,4,0)** would produce a diamond shape, ie a square rotated so that its diagonals are vertical and horizontal.

PROCstar(ri%,ro%,points%,as) produces a star, with **points%** specifying the number of points, working from an inner radius **ri%,** to an outer radius **ro%;** the starting angle specified by **as.** Thus for a spikey star, **ri%** should be small compared with **ro%.**

PROCinitialise sets all the default values of variables and also programs a couple of useful keys. Key 1 lists the program up to, but not including, the procedures; ie it lists the bit of the program you've written yourself. Key 2 calls **PROChome** to move the arm off the paper.

Listings 2 and 3 are fairly straightforward and should illustrate techniques by which patterns can be gnerated. **Listing 4** shows just how simple a program can be to produce some very interesting effects.

Listing 5 should be typed in and run if you want to generate the machine code program needed for running the plotter through the RS423 interface. The connections for the lead are shown in **Figure 4.**

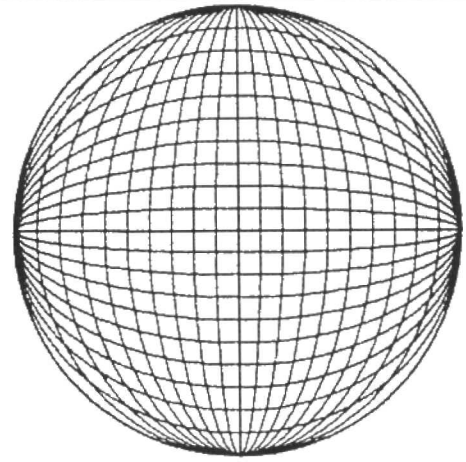
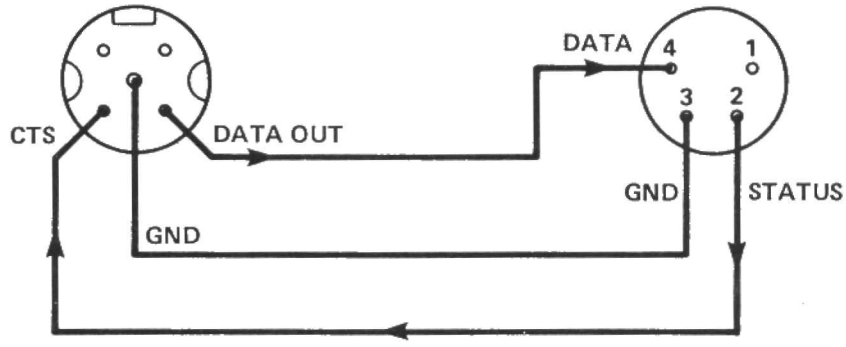


Figure 3 (right). Display produced by Listing 4.
Figure 4 (below). Connection diagram for RS423-RS232 cable.



(View shown looking into back of plug ie; looking into the socket)

LISTING 5. Source Code for machine code RS423-RS232 driver.

```
 10 PRINT "TAPE or DISC (T/D)?"
 20 G=GET
 30 IF G=ASC"T" M%=&D01 ELSE M%=&A00
 40
 50 WRCHVEC=&20E
 60 OSBYTE=&FFF4
 70
 80 FOR opt=0 TO 2 STEP 2
 90   P%=M%
100   [OPTopt
110
120   .start
130   LDA #8
140   LDX #4
150   JSR OSBYTE
160   LDA #156
170   LDX #16
180   LDY #227
190   JSR OSBYTE
200   LDA #5
210   LDX #2
220   JSR OSBYTE
230
240   LDA WRCHVEC
250   STA oldWRCHVEC
260   LDA #newWRCHVEC MOD 256
270   STA WRCHVEC
280   LDA WRCHVEC + 1
290   STA oldWRCHVEC + 1
300   LDA #newWRCHVEC DIV 256
310   STA WRCHVEC + 1
320   RTS
330
340   .newWRCHVEC
350   STA Astore
360   STX Xstore
370   STY Ystore
380
390   LDA #&75
400   JSR OSBYTE
410   TXA
420   AND #1
430   BEQ out
440
450   .checkACIA
460   LDA #&81
470   LDX #&8F
480   LDY #&FF
490   JSR &FFF4
500   CPY #0
510   BNE out
520   LDA #2
530   BIT &FE08
540   BEQ checkACIA
550
560   LDA #3
570   STA count
580   .wait
590   DEC count+1
600   BNE wait
610   DEC count
620   BNE wait
630
640   .out
650   LDA Astore
660   LDX Xstore
670   LDY Ystore
680   JMP (oldWRCHVEC)
690
700   .Astore
710   NOP
720   .Xstore
730   NOP
740   .Ystore
750   NOP
760   .oldWRCHVEC
770   NOP:NOP
780   .count
790   NOP
800   ]
810   NEXT
820
830 PRINT" *SAVE A4 ";~start" ";~P%" ";~start
840 CALL start
```

TIME TRACK

SPLIT SECOND TIMING

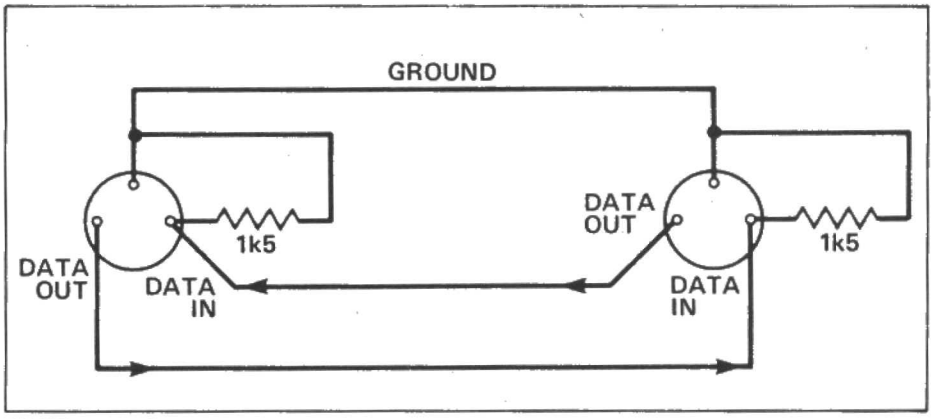
Paul Beverley had developed a software link to the RS423 interface which lays down a time track on audio or video tapes, allowing split-second, precision editing.

High quality editing of multi-track recording tape requires accurate timing. This is an ideal application for a computer, which can lay time data on to one of the tracks; it is then possible to pin-point other parts of the tape with split-second timing by referring to the time track – even after cutting and splicing. In the May issue of *E&CM* we looked at a technique of using the BBC micro to send bytes of information, in real time, via the cassette interface by making a software link with the RS423 interface. Exactly this technique can be used to lay down a time track.

The same method can be applied to scientific data recording where again you may have a multi-track recorder pulling in data from various recording devices and need to pin-point the exact time at which an event took place. The movement of the tape could be timed, but the accuracy would be limited by the accuracy of the speed of the recording machine. However, if you lay down a timing track then even if the tape became stretched or the speed of playback was slightly different from the recording speed, you would still have an accurate indication of timing as given by the time track.

To generate this time track, all you need to do is modify the program given in the May issue to send out, at fixed time intervals, bytes of information giving the current time as registered by the TIME value.

There is virtually no hardware needed for this system since the computer itself generates the tones which carry the information. All you need is an appropriate lead to transfer the signals between the DIN plug of the cassette interface and the multi-track recorder that you are using. The exact configuration of the system is decided by the software; the programs listed in this article give two possible ways of setting things up.



RS423 connections.

Timing in minutes and seconds

The first program, **Listing 1,** is written entirely in BASIC. It provides an indication of time in minutes and seconds and is fairly straightforward in operation. The program contains all the procedures needed both for recording the track and also replaying it, but because it is written in procedures, it shouldn't be too difficult to split it up into two separate programs for recording and replay. The time track is actually laid down in seconds but in the replay routine this is printed out in minutes and seconds.

The operation of the program is as follows. When you select the recording mode then once every second, as indicated by the TIME value, the current time value in seconds (TIME DIV 100) is output as a string to the RS423 channel. This has previously been linked to the cassette interface by a series of FX calls which were explained in the original article in the May issue. The characters of the string are transmitted as audible tones. On playback, the program waits for bytes to come in on the RS423 (and hence from the cassette interface). These bytes are tested, and only the numbers 0 to 9 are accepted. (This is necessary because by picking up the time track at different points, you can get erroneous characters.) Finally the display is generated giving the last received time value in minutes and seconds.

Timing in tenths

To increase the resolution of the timing, say to one tenth of a second, you have to consider the maximum rate at which data can be put onto the tape. The rate is limited by the data rate of 300 baud. You should use 1200 baud, but in the trials that I have done this didn't prove sufficiently reliable. 300 baud represents 300 bits per second and therefore only 30 bits in 0.1 seconds. This means that you cannot send more than three 8 bit bytes in each time interval. Unfortunately you really need to use one byte to synchronise the transmission leaving only 2 bytes to specify the data. Thus you can record intervals of 0.1 seconds for a total of 65535 x 0.1 seconds, which is about 10 minutes. Beyond that time the time indication goes back to zero and starts again. Thus if you want to know the

length of time between, say, the beginning of the tape and a point which is 15 minutes into the tape, you would have to add 6553.5 seconds to the time indicated.

To go beyond 10 minutes and still be able to indicate the total time elapsed, there are two courses of action. The first is to ignore the fact that you cannot fit 4 bytes into the time interval and allow the system to keep going regardless. The effect of this is that occasionally a time indication is missed out as the buffer in the computer becomes full and the next time indicator is lost. This may cause slight inaccuracies as the buffer fills and is allowed to empty slightly before the next time value is accepted, but the error would never be more than about 0.2 seconds. **Listing 2** will operate in this manner, as it stands, doing its best to give 3 bytes of time data each tenth of a second. To reduce the number of time bytes to 2, simply omit lines 1400, 1410, 1580 and 1590.

The other alternative is to use 3 bytes of time data, sent out only every 0.2 seconds. This can be done by changing 5 into 10 at lines 1170 and 1420.

In order to send and receive bytes of information at the speed required, it is not possible to use BASIC throughout. Some explanation is needed as to how this program actually works.

To record the data an event routine, written in machine code (1150 – 1220), counts the vertical sync pulses which occur every 0.02 seconds. After five pulses have been received a flag is set to indicate the fact. Meanwhile the "send" routine (1240 – 1470), again in machine code, is sitting waiting till this flag is set. It waits until the countdown reaches 1 and then reads the TIME value using the OSWORD routine, putting the result at location &CF0 onwards. A synchronising byte (&AA) is sent out, followed by either 2 or 3 bytes of the time value. Then, to avoid sending out another set of time bytes during the same time interval, you sit and wait until the count has begun again, indicated by the fact that the flag at &CFE has been re-set to 5.

The reading routine (1490 – 1600) is much more simple and consists simply of waiting until a sync character arrives and then pulling in the next 2 or 3 bytes, in each case using the OSRDCH routine, and storing them in the bytes of the integer variable, A%. The BASIC part of the replay routine (500 – 720) is similar to the previous program except that it enables the RS423 as input (620), calls the machine code routine to get the next time byte (630) and then switches back to receive from the keyboard (640). This time value is then printed out in seconds and tenths (650 – 670) and the keyboard is checked to see whether the user is trying to escape from the replay routine.

Applications

Thanks to Ronnie Boyd for this very interesting idea. There may well be other RS423/cassette applications which you can think of but don't feel able to tackle; in which case, let us know and we will try to develop the software and publish it for the benefit of other readers.

LISTING 1. Time-track recording minutes/seconds.

```
 10 REM Time-track recording & replay
 20 REM via the cassette interface.
 30 REM Display in minutes and seconds.
 40 REM(C)1985 Norwich Computer Services
 50
 60 MODE 7
 70 PROCinitialise
 80 REPEAT
 90   PROCselect_mode
100   IF record PROCrecord
110   IF replay PROCreplay
120   UNTIL quit=TRUE
130 PROCtidy_up
140 END
150
160 DEF PROCselect_mode
170 record=FALSE
180 replay=FALSE
190 PRINT"Record, Play-back or Quit? (R/P/Q)";
200 R$=GET$
210 PRINT R$
220 IF (ASC(R$)AND95)=80 replay=TRUE
230 IF (ASC(R$)AND95)=81 quit=TRUE
240 IF (ASC(R$)AND95)=82 record=TRUE
250 ENDPROC
260
270 DEF PROCrecord
280 *MO.0
290 *FX138,1,1
300 *FX138,1,1
310 *FX2,2
320 *FX8,3
330 *FX156,3,252
340 *FX156,2,252
350 *FX203,9
360 *FX3,5
370 PRINT '"Press space bar to start recording."
380 PRINT '"Press space bar to stop."
390 start=GET
400 TIME=0
410 wait=INKEY50
420 REPEAT
430   REPEAT UNTIL TIME MOD100=0
440   PRINT TIME DIV 100
450   UNTIL INKEY(0)=32
460 REPEAT
470   UNTIL ADVAL(-3)>&8E
480 TIME=0
490 REPEAT
500   UNTIL TIME=50
510 *FX3,0
520 *FX203,255
530 ENDPROC
540
550 DEF PROCreplay
560 *MO.1
570 *FX7,3
580 *FX156,3,252
590 *FX156,2,252
600 *FX21,1
610 *FX2,1
620 CLS
630 PRINT TAB(0,22)"Press ESCAPE to stop."
640 PRINT TAB(H%-1,V%)CHR$(141)
650 PRINT TAB(H%-1,V%+1)CHR$(141)
660 REPEAT
670   T$=""
680   REPEAT
690     N%=GET
700     IF N%>47 AND N%<58 T$=T$+CHR$(N%)
710     UNTIL N%=13
720   T%=EVAL(T$)
730   M%=T%DIV60
740   S$=STR$(T%MOD60)
750   IF LEN(S$)=1 S$="0"+S$
760   PRINT TAB(H%,V%);M%;".";S$;"    "
770   PRINT TAB(H%,V%+1);M%;".";S$;"    "
780   UNTIL 0
790 *FX2,2
800 ENDPROC
810
820 DEF PROCinitialise
830 ON ERROR PROCerror:END
840 *FX205,64
850 quit=FALSE
860 at%=0%
870 @%=1
880 H%=10:V%=10
890 ENDPROC
900
910 DEF PROCerror
920 *FX205,0
930 *FX2,0
940 *FX3,0
950 IF ERR=17 RUN
960 REPORT:PRINT" at line ";ERL
970 *MO.0
980 ENDPROC
990
1000 DEF PROCtidy_up
1010 *FX205,0
1020 *FX2,0
1030 *FX3,0
1040 *MO.0
1050 CLS
1060 @%=at%
1070 ENDPROC
```

LISTING 2. Time-track recording in tenths.

```
 10 REM Time-track recording & replay
 20 REM via the cassette interface.
 30 REM Display in tenths of seconds.
 40 REM(C)1985 Norwich Computer Services
 50
 60 MODE 7
 70 PROCinitialise
 80 PROCassemble
 90 REPEAT
100   PROCselect_mode
110   IF record PROCrecord
120   IF replay PROCreplay
130   UNTIL quit=TRUE
140 MODE 7
150 PROCtidy_up
160 END
170
180 DEF PROCselect_mode
190 CLS
200 record=FALSE
210 replay=FALSE
220 PRINT"Record, Play-back or Quit? (R/P/Q)";
230 R$=GET$
240 PRINT R$
250 IF (ASC(R$)AND95)=80 replay=TRUE
260 IF (ASC(R$)AND95)=81 quit=TRUE
270 IF (ASC(R$)AND95)=82 record=TRUE
280 IF ASC(R$)=32 replay=TRUE
290 ENDPROC
300
310 DEF PROCrecord
320 *MO.0
330 *FX138,1,1
340 *FX138,1,1
350 *FX2,2
360 *FX8,3
370 *FX156,3,252
380 *FX156,2,252
390 *FX203,9
400 PRINT '"Press space bar to start recording"
410 PRINT '"Press BREAK to stop"
420 start=GET
430 TIME=0
440 *FX3,7
450 *FX14,4
460 CALL send
470 ENDPROC
480
490 DEF PROCreplay
500 *MO.1
510 *FX7,3
520 *FX156,3,252
530 *FX156,2,252
540 *FX21,1
550 CLS
560 PRINT TAB(0,22)"Press space bar to pause"
570 PRINT TAB(0,24)"Press return to stop"
580 PRINT TAB(H%-1,V%)CHR$(141)
590 PRINT TAB(H%-1,V%+1)CHR$(141)
600 REPEAT
610   REPEAT
620     *FX2,1
630     CALL get
640     *FX2,2
650     T$=STR$(A%DIV10)+"      "
660     PRINT TAB(H%,V%)T$
670     PRINT TAB(H%,V%+1)T$
680     K%=INKEY0
690     UNTIL K%>0
700   IF K%=32 PROChold
710   UNTIL K%=13
720 ENDPROC
730
740 DEF PROChold
750 *FX2,0
760 PRINT TAB(0,V%+2)"HOLD"
770 REPEAT UNTIL GET=32
780 PRINT TAB(0,V%+2)"    "
790 ENDPROC
800
810 DEF PROCinitialise
820 ON ERROR MODE 7:PROCerror:END
830 *KEY10OLD MRUN M
840 *FX205,64
850 VDU23;8202;0;0;0;
860 quit=FALSE
870 H%=10:V%=10
880 ENDPROC
890
900 DEF PROCerror
910 *FX203,255
920 *FX205,0
930 *FX2,0
940 *FX3,0
950 *FX13,4
960 REPORT:PRINT" at line ";ERL
970 *MO.0
980 ENDPROC
990
1000 DEF PROCtidy_up
1010 *FX205,0
1020 *FX2,0
1030 *FX3,0
1040 *MO.0
1050 ENDPROC
1060
1070 DEF PROCassemble
1080 OSWORD=&FFF1
1090 OSWRCH=&FFEE
1100 OSRDCH=&FFE0
1110 FOR opt%=0TO2STEP2
1120   P%=&C00
1130   [OPT opt%
1140
1150   DEC &CFF    \ Divide-by-5 counter.
1160   BNE out     \ Not zero? Then give up.
1170   LDA #5      \ Reset it to five ready
1180   STA &CFF    \ to count down again.
1190   STA &CFE    \ Flag to avoid bytes being
1200               \ sent twice each 1/10 sec.
1210   .out
1220   RTS
1230
1240   .send
1250   LDA &CFF    \ Wait for a given point in the
1260   CMP #1      \ 1/10 second cycle,
1270   BNE send
1280
1290   STA &CFE    \ Set flag to show bytes going.
1300   LDA #1      \ OSWORD 1 = Read the TIME value
1310   LDX #&F0    \ to location &CF0.
1320   LDY #&C
1330   JSR OSWORD
1340   LDA #&AA    \ Send a synchronising byte.
1350   JSR OSWRCH
1360   LDA &CF0    \ Send low byte of TIME value.
1370   JSR OSWRCH
1380   LDA &CF1    \ Send next byte of TIME value.
1390   JSR OSWRCH
1400   LDA &CF2    \ Send third byte of TIME value.
1410   JSR OSWRCH
1420   LDA #5      \ Ready to check flag.
1430
1440   .next
1450   CMP &CFE    \ Wait till flag = 5.
1460   BNE next    \ i.e. don't send
                   another set during
1470   JMP send    \ this 1/10 second interval.
1480
1490   .get
1500   JSR OSRDCH \ Wait until a sync. character
1510   CMP #&AA   \ arrives.
1520   BNE get
1530
1540   JSR OSRDCH \ Get low byte.
1550   STA &404   \ Put it in low byte of A%
1560   JSR OSRDCH \ Get next byte.
1570   STA &405   \ Put it in next byte of A%
1580   JSR OSRDCH \ Get third byte.
1590   STA &406   \ Put it in third byte of A%
1600   RTS
1610   ]
1620   NEXT
1630 ?&220=0 :REM set event vector
1640 ?&221=&C
1650 ENDPROC
```

QL CAD

Angus McFadzean has written a computer aided circuit design program for the QL, with a library of electronic symbols and a wealth of user commands.

Some engineers may still prefer to draw circuit diagrams on the back of a paper napkin, but most use sophisticated computer aided design (CAD) systems. QCAD is a computer aided drawing program for the QL which enables the amateur (or professional) designer to draw, save and print out circuit diagrams using a library of component symbols.

The program, although written in BASIC, is surprisingly fast and easy to use. Virtually all of the commands are 'idiot-proofed' to prevent any accidental key presses causing disasterous results. This in part accounts for the length of the program, the main reason being the number of in-built commands available to the user.

The program is shown in **Listing 1** with a sample print-out in **Figure 1.** Several points should be made about the program.

First the screen is copied to a printer by using the routine supplied with EASEL. Memory for this routine is reserved at line 150 and the program then loaded. It should be noted that the routine is assumed to be on drive one and that the program name is 'gprint_prt'. However this program may have a different name on different versions of EASEL and should be changed to whatever is appropriate.

Also worth noting is that the program was written on a TV set. It should therefore work on all monitors, but the windows were expanded from normal TV size to completely fill the screen, and therefore give as large a drawing area as possible. For this reason some TV users may have to reduce the window sizes, and some monitor users may wish to expand them.

Some engineers may still prefer to draw circuit diagrams on the back of a paper napkin – most use computer aided design.

When run, the program will produce a blank graphics area except for the cursor in the middle. Above the graphics area is the prompts window. Any resemblance to other prompt windows, living or dead, is purely coincidental. The prompts area tells you which key to press to draw the predefined components and also what the function keys do. This area also displays any additional information required after a command has been entered.

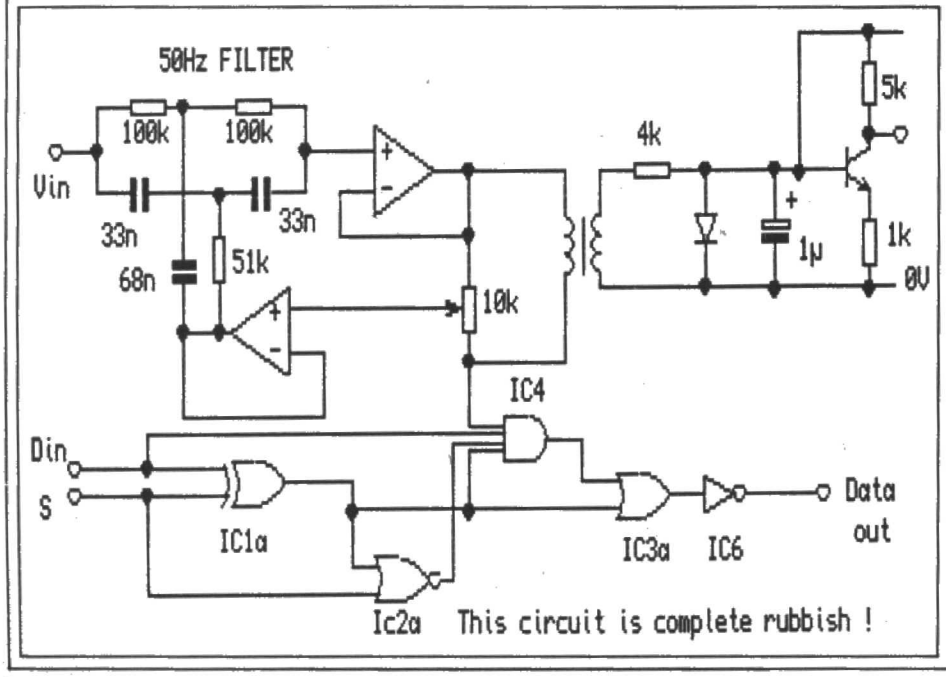
Upon start up, the program is in move mode, ie using the cursor keys to move the cursor will simply move it without leaving a line. To draw a line, line mode must be entered and this is accomplished by pressing F1. To revert to move mode simply press F2.

The cursor will move ten pixels at a time when any of the cursor keys are pressed. However, this can be slowed down to one pixel at a time by simultaneously pressing CTRL and a cursor key, or speeded up to thirty at a time by simultaneously pressing ALT and a cursor key. This makes it possible to draw very fine detail but also enables the cursor to be moved around the screen relatively quickly.

Also controlled by the function keys are the commands2 and erase modes. These are entered by pressing F3 and F4 respectively. Commands2 mode updates the prompts window to display more predefined components and some extra commands which will be described later. Pressing F3 when in commands2 mode will cause commands1 mode to be entered once more.

Erase mode allows the user to delete a block of the graphics window by making the current cursor position one corner of a rectangle and then asking the cursor to be updated to the diagonally opposite corner of the rectangle. All the contents of this rectangle are then deleted. The rectangle may be made as large or as small as wished, so it is possible to delete very small mistakes in the middle of fine detail.

Turning now to the pre-defined components, it can be seen that there are a large number available. In commands1 mode



resistors, capacitors (electrolitic or non-electrolitic), inductors, diodes, op-amps, transistors, AND, NAND, OR, NOR and Exclusive-OR gates, an earth symbol, a small circle – to make up switches, connectors etc. and finally a connection circle to place where two lines join. Commands2 mode offers arrows, edge connectors, and buffer and inverter gates. Also available in this mode are the commands to print, load or save a drawing, a command to write text anywhere on the screen, and also commands to clear the screen, change the ink colour or draw diagonal lines.

The program, although written in BASIC, is surprisingly fast and easy to use.

When a pre-defined component is chosen, a direction, if relevant, is asked for. Thus, for example, if you have just drawn a line using the right cursor, the direction required for the component is right, ie the component is drawn going to the right of the screen. All the components that are directional can be drawn either up, down, left, or right, except for the op-amp. It was felt that this would only ever be drawn in

QL CAD

```
10 REMark ******************************
20 REMark *
30 REMark *       CAD for the QL
50 REMark *     by Angus McFadzean
55 REMark *       (C) April 1985
60 REMark *
70 REMark ******************************
80 REMark


100 MODE 4: PAPER 0: cur_col=4: INK 4: S
CALE 464,0,0: DIM c(7)
110 WINDOW #1,464,200,25,53: CLS
120 OPEN #5,con_: OPEN #4,scr_
130 WINDOW #4,464,43,25,10: WINDOW #5,31
2,43,84,10
140 prompt$: x=262: y=300: d_mode=2: LIN
E x,y: cross: menu
150 mem=RESPR(2048): LBYTES mdv1_gprint_
prt, mem
160 REPeat key_sel
170     get_key
180     SELect ON k_code
190         =232: d_mode=1
200         =236: d_mode=2
210         =240: IF com_flag=1: menu2: EL
SE menu
220         =244: erase: cross: menu
230         =REMAINDER :
240     END SELect
250     cursor_keys
260     IF com_flag=1
270         SELect ON k_code
280             =82: res
290             =67: pos1=1: pos2=1: cap
300             =73: ind
310             =68: diode
320             =65: and
330             =78: and: LINE x,y: dr_circ
340             =79: or
350             =75: or: LINE x,y: dr_circ
360             =80: op_amp
370             =84: trans
380             =83: elec_cap
390             =88: xor
400             =81: dirc(4): dr_circ
410             =69: earth
420             =74: connec
430             =REMAINDER : NEXT key_sel
440         END SELect
450     ELSE
460         SELect ON k_code
470             =65: arrow
480             =69: Edge_con
490             =84: Text
500             =80: printer
510             =83: msave
520             =76: mload
530             =90: zap
540             =73: buffer: LINE x,y: dr_c
irc
550             =67: ink_col
560             =68: diagonal
570             =66: buffer
580             =REMAINDER : NEXT key_sel
590         END SELect
600     END IF
610     menu: cross
620 END REPeat key_sel
630 DEFine PROCedure res
640     dirc(4)
650     SELect ON dir_c
660         =0:RETurn
670         =1: LINE_R 5,0 TO 0,30 TO -10,
0 TO 0,-30 TO 10,0: y=y+30
680         =2: LINE_R 0,5 TO 30,0 TO 0,-1
0 TO -30,0 TO 0,10: x=x+30
690         =3: LINE_R 5,0 TO 0,-30 TO -10
,0 TO 0,30 TO 10,0: y=y-30
700         =4: LINE_R 0,5 TO -30,0 TO 0,-
10 TO 30,0 TO 0,10: x=x-30
710     END SELect
720 END DEFine
730 DEFine PROCedure cap
740     dirc(4)
750     SELect ON dir_c
760         =0: RETurn
770         =1: LINE_R -10,5: plate_ud(pos
1)
780             LINE_R 0,10: plate_ud(pos2
): y=y+15
790         =2: LINE_R 0,10: plate_lr(pos1
)
800             LINE_R 10,0: plate_lr(pos2
): x=x+15
810         =3: LINE_R -10,0: plate_ud(pos
1)
820             LINE_R 0,-10: plate_ud(pos
2): y=y-15
830         =4: LINE_R -5,10: plate_lr(pos
1)
840             LINE_R -10,0: plate_lr(pos
2): x=x-15
850     END SELect
860 END DEFine
870 DEFine PROCedure plate_lr (pos1)
880     FILL pos1: LINE_R TO 5,0 TO 0,-21
 TO -5,0 TO 0,21: FILL 0
890 END DEFine
900 DEFine PROCedure plate_ud(pos1)
910     FILL pos1: LINE_R TO 21,0 TO 0,-5
 TO-21,0 TO 0,5: FILL 0
920 END DEFine
930 DEFine PROCedure ind
940     dirc(4)
950     SELect ON dir_c
960         =0: RETurn
970         =1: FOR i=1 TO 3: ARC_R TO 0,1
3,PI: NEXT i: y=y+39
980         =2: FOR i=1 TO 3: ARC_R TO 13,
0,-PI: NEXT i: x=x+39
990         =3: FOR i=1 TO 3: ARC_R TO 0,-
13,PI:NEXT i: y=y-39
1000        =4: FOR i=1 TO 3: ARC_R TO -1
3,0,PI:NEXT i: x=x-39
1010    END SELect
1020 END DEFine
1030 DEFine PROCedure diode
1040    dirc(4)
1050    SELect ON dir_c
1060        =0: RETurn
1070        =1: LINE_R 9,0 TO -18,0 TO 9,
20 TO 9,-20; 1,20 TO -20,0: y=y+20
1080        =2: LINE_R 0,-9 TO 0,18 TO 20
,-9 TO -20,-9; 20,0 TO 0,20: x=x+20
1090        =3: LINE_R -9,0 TO 18,0 TO -9
,-20 TO -9,20; 0,-20 TO 20,0: y=y-20
1100        =4: LINE_R 0,-9 TO 0,18 TO -2
0,-9 TO 20,-9; -20,0 TO 0,20: x=x-20
1110    END SELect
1120 END DEFine
1130 DEFine FuNction answer(ques$)
1140    REPeat reply_in
1150        CLS #5: PRINT #5;ques$;' Y or
 N'
1160        get_key
1170        SELect ON k_code
1180            =89: RETurn 1
1190            =78: RETurn 0
1200            =27: RETurn -1
1210            =REMAINDER : NEXT reply_in
1220        END SELect
1230    END REPeat reply_in
1240 END DEFine
1250 DEFine PROCedure op_amp
1260    dirc(2): IF dir_c=0: RETurn
1270    qu$=' Start drawing from top ter
minal ?': ans=answer(qu$)
1280    SELect ON ans
1290        =-1: RETurn
1300        =1: LINE_R 0,15: y=y-15
1310        =0: LINE_R 0,45: y=y+15
1320    END SELect
1330    SELect ON dir_c
1340        =2: di=1: msk=15
1350        =4: di=-1: msk=0
1360    END SELect
1370    LINE_R TO di*50,-30 TO -di*50,-3
0 TO 0,60
1380    LINE_R 0,-15 TO -di*10,0: 0,-30
TO di*10,0: x=x+di*50
1390    ques$=' Is top terminal invertin
g ?': ans=answer(ques$)
1400    IF ans=-1: RETurn : ELSE CURSOR
x-(12&msk),y+25,-di*19,0
1410    OVER -1: IF ans: PRINT;'-': s=1:
 ELSE PRINT;'+': s=0
1420    CURSOR x-(12&msk),y,-di*19,0: I
F s=1: PRINT;'+': ELSE PRINT;'-'
1430 END DEFine
1440 DEFine PROCedure elec_cap
1450    qu$=' Is nearest plate to cursor
 positive ?'
1460    ans=answer(qu$): IF ans=-1:cross:
RETurn :ELSE pos2=ans:pos1=NOT pos2
1470    cap
1480    SELect ON dir_c
1490        =0: RETurn
1500        =1: IF pos1: ofx=5: ofy=-11:
ELSE ofx=5: ofy=6
1510        =2: IF pos1: ofx=5: ofy=0: EL
SE ofx=-20: ofy=0
1520        =3: IF pos2: ofx=6: ofy=-17:
ELSE ofx=5: ofy=0
1530        =4: IF pos2: ofx=12: ofy=0: E
LSE ofx=-10: ofy=0
1540    END SELect
1550    CURSOR x,y,ofx,ofy: PRINT;'+'
1560 END DEFine
1570 DEFine PROCedure get_key
1580    REPeat key: k$=INKEY$: IF k$<>''
 THEN EXIT key
1590    k_code=CODE(k$): IF k_code<123 A
ND k_code>60: k_code=k_code&&223
1600 END DEFine
1610 DEFine PROCedure dirc(no)
1620    cross
1630    REPeat sel_dir
1640        CLS #5: IF no=4
1650                PRINT #5;' Up, dow
n, left or right ?'
1660            ELSE
1670                PRINT #5;' Left or
 right ?'
1680            END IF
1690        get_key
1700        SELect ON k_code
1710            =27: dir_c=0
1720            =85: dir_c=1
1730            =82: dir_c=2
1740            =68: dir_c=3
1750            =76: dir_c=4
1760            =REMAINDER : NEXT sel_dir
1770        END SELect
1780        EXIT sel_dir
1790    END REPeat sel_dir
1800 END DEFine
1810 DEFine PROCedure menu
1820    com_flag=1: CURSOR #4,372,5: PRI
NT #4;'F3 - COMMANDS2'
1830    CLS #5: INK #5,7: PRINT #5;' COM
MANDS1';
1840    INK #5,4: PRINT #5;'   AND  ..A
   Elec cap .S    Connec .J'
1850    PRINT #5;' Res ..R     NAND ..N
   Op-amp  ..P    Diode ..D'
1860    PRINT #5;' Cap ..C     OR   ..O
   Trans   ..T    Circle .Q'
1870    PRINT #5;' Ind ..I     NOR  ..K
   XOR     ..X    Earth ..E'
1880 END DEFine
1890 DEFine PROCedure menu2
1900    com_flag=2: CURSOR #4,372,5: PRI
NT #4;'F3 - COMMANDS1'
1910    CLS #5: INK #5,7: PRINT #5;' COM
MANDS2';
1920    INK #5,4: PRINT #5;'   Text  ..T
   Edge con ..E'
1930    PRINT #5;' Print .P    Zap   ..Z
   Inverter ..I'
1940    PRINT #5;' Save ..S    Arrow ..A
   Diagonal ..D'
1950    PRINT #5;' Load ..L    Colour..C
   Buffer   ..B'
1960 END DEFine
1970 DEFine PROCedure c_move
1980    SELect ON d_mode
1990        =1: LINE xo,yo TO x,y
2000        =2: LINE x,y
2010    END SELect
2020    cross
2030 END DEFine
2040 DEFine PROCedure up(disp)
2050    cross: xo=x: yo=y: y=y+disp
2060    c_move
2070 END DEFine
2080 DEFine PROCedure down(disp)
2090    cross: xo=x: yo=y: y=y-disp
2100    c_move
2110 END DEFine
2120 DEFine PROCedure right(disp)
2130    cross: xo=x: yo=y: x=x+disp
2140    c_move
2150 END DEFine
2160 DEFine PROCedure left(disp)
2170    cross: xo=x: yo=y: x=x-disp
2180    c_move
2190 END DEFine
2200 DEFine PROCedure cross
2210    OVER -1
2220    LINE x,y: LINE_R 0,10 TO 0,-21;
-10,11 TO 20,0: LINE x,y
2230    OVER 0
2240 END DEFine
2250 DEFine PROCedure connec
2260    cross
2270    FILL 1: CIRCLE_R 0,0,5: FILL 0
2280 END DEFine
2290 DEFine PROCedure dr_circ
2300    SELect ON dir_c
```

QL CAD (Continued)

```
2310        =0: RETurn
2320        =1: CIRCLE_R 0,5,5: y=y+10
2330        =2: CIRCLE_R 5,0,5: x=x+10
2340        =3: CIRCLE_R 0,-5,5: y=y-10
2350        =4: CIRCLE_R -5,0,5: x=x-10
2360    END SELect
2370 END DEFine
2380 DEFine PROCedure earth
2390    cross: LINE_R 12,0 TO -24,0; 5,-
5 TO 17,0; -5,-5 TO -8,0
2400 END DEFine
2410 DEFine PROCedure zap
2420    cross: CLS #5: PRINT #5;' Type Y
 to confirm'
2430    get_key: IF k_code=89 THEN CLS
2440 END DEFine
2450 DEFine PROCedure Text
2460    CLS #5: PRINT #5;' Enter text an
d press enter:'
2470    CURSOR x,y,0,-10
2480    cross: INPUT text$: x=x+(12*LEN(
text$))
2490 END DEFine
2500 DEFine PROCedure gate
2510    dirc(4): IF dir_c=0: RETurn
2520    PRINT #5;' Number of inputs (2 o
r 4) ?': get_no
2530    IF k_code=27: dir_c=0: RETurn :
ELSE ip_no=k_code-48
2540    PRINT #5;' Which input to start
 drawing from ?': get_no
2550    IF k_code=27: dir_c=0: RETurn :
ELSE st_no=k_code-48
2560    SELect ON dir_c
2570      =1,3: SELect ON st_no
2580          =1: LINE_R -6,0: x=x
+9

2590          =2: IF ip_no=2:LINE_
R -24,0:x=x-9:ELSE LINE_R -12,0:x=x+3
2600          =3: LINE_R -18,0: x=
x-3
2610          =4: LINE_R -24,0: x=
x-9
2620        END SELect
2630        IF dir_c=1: difl=1: ELS
E difl=-1
2640      =2,4: SELect ON st_no
2650          =1: LINE_R 0,6: y=y-
9
2660          =2: IF ip_no=2:LINE_
R 0,24: y=y+9:ELSE LINE_R 0,12: y=y-3
2670          =3: LINE_R 0,18: y=y
+3
2680          =4: LINE_R 0,24: y=y
+9
2690        END SELect
2700        IF dir_c=2: difl=1: ELS
E difl=-1
2710    END SELect
2720 END DEFine
2730 DEFine PROCedure and
2740    gate
2750    SELect ON dir_c
2760      =1,3: and_ud ip_no,difl: y=y+
difl*35
2770      =0: RETurn
2780      =2,4: and_rl ip_no,difl: x=x+
difl*35
2790    END SELect
2800 END DEFine
2810 DEFine PROCedure and_rl(ip_no,di)
2820    LINE_R TO di*20,0: ARC_R TO 0,-3
0,-di*PI
2830    LINE_R TO -di*20,0 TO 0,30: 0,-6
 TO -di*10,0: 0,-18 TO di*10,0
2840    IF ip_no=4: LINE_R 0,6 TO -di*10
,0: 0,6 TO di*10,0
2850 END DEFine
2860 DEFine PROCedure and_ud(ip_no,di)
2870    LINE_R TO 0,di*20: ARC_R TO 30,0
,-di*PI
2880    LINE_R TO 0,-di*20 TO -30,0: 6,0
 TO 0,-di*10: 18,0 TO 0,di*10
2890    IF ip_no=4: LINE_R -6,0 TO 0,-di
*10: -6,0 TO 0,di*10
2900 END DEFine
2910 DEFine PROCedure or
2920    gate
2930    SELect ON dir_c
2940      =1,3: or_ud difl: ip_ud ip_no
,difl: y=y+difl*45
2950      =2,4: or_rl difl: ip_rl ip_no
,difl: x=x+difl*45
2960    END SELect
2970 END DEFine
2980 DEFine PROCedure or_rl(di)
2990    LINE_R TO di*20,0: ARC_R TO di*2
5,-15,-di*PI/4 TO -di*25,-15,-di*PI/4
3000    LINE_R TO -di*20,0: ARC_R TO 0,3
0,di*PI/2.5
3010 END DEFine
3020 DEFine PROCedure ip_rl (ip_no,di)
3030    LINE_R di*3,-6 TO -di*10,0; 0,-1
8 TO di*10,0
3040    IF ip_no=4: LINE_R di,6 TO -di*1
0,0; 0,6 TO di*10,0
3050 END DEFine
3060 DEFine PROCedure or_ud (di)
3070    LINE_R TO 0,di*20: ARC_R TO 15,d
i*25,-di*PI/4 TO 15,-di*25,-di*PI/4
3080    LINE_R TO 0,-di*20: ARC_R TO -30
,0,di*PI/2.5
3090 END DEFine
3100 DEFine PROCedure ip_ud (ip_no,di)
3110    LINE_R 6,di*3 TO 0,-di*10; 18,0
TO 0,di*10
3120    IF ip_no=4: LINE_R -6,di TO 0,-d
i*10; -6,0 TO 0,di*10
3130 END DEFine
3140 DEFine PROCedure xor
3150    gate
3160    SELect ON dir_c
3170      =0: RETurn
3180      =1,3: ARC_R TO 30,0,-difl*PI/
2.5: LINE_R -30,difl*8
3190          or_ud difl: LINE_R 0,-d
ifl*10
3200          ip_ud ip_no,difl: y=y+d
ifl*51
3210      =2,4: ARC_R TO 0,-30,-difl*PI
/2.5: LINE_R difl*8,30
3220    IF type=-1: RETurn
3230          or_rl difl: LINE_R -dif
l*10,0
3240          ip_rl ip_no,-difl: x=x+
difl*51
3250    END SELect
3260 END DEFine
3270 DEFine PROCedure cursor_keys
3280    SELect ON k_code
3290      =208: up(10)
3300      =216: down(10)
3310      =192: left(10)
3320      =200: right(10)
3330      =210: up(1)
3340      =218: down(1)
3350      =194: left(1)
3360      =202: right(1)
3370      =193: left(30)
3380      =201: right(30)
3390      =209: up(30)
3400      =217: down(30)
3410      =REMAINDER :
3420    END SELect
3430 END DEFine
3440 DEFine PROCedure erase
3450    CLS#5: PRINT #5;' Position curso
r to diagonal corner of rectangle'
3460    PRINT #5;' that is to be erased
and press enter.'
3470    sec_point: cross: INK 0
3480    FILL 1: LINE x,y TO xold,y TO xo
ld,yold TO x,yold TO x,y: FILL 0
3490    INK cur_col: x=xold: y=yold: d_m
ode=d_model
3500 END DEFine
3510 DEFine PROCedure sec_point
3520    d_model=d_mode: d_mode=2: xold=x
: yold=y

3530    REPeat move_c
3540      get_key: cursor_keys
3550      IF k_code=10 THEN EXIT move_c
3560    END REPeat move_c
3570 END DEFine
3580 DEFine PROCedure ink_col
3590    FOR i=2 TO 7: c(i)=i
3600    cross: CLS #5: PRINT #5;' Enter
new ink colour number.'
3610    get_no: IF k_code=27: RETurn : E
LSE col=k_code-48
3620    SELect ON col
3630      =0,1,3,5,6: RETurn
3640      =2,4,7: c(cur_col)=col: INK c
ol: cur_col=col: c(6)=c(7)
3650    END SELect
3660    RECOL c(0),c(1),c(2),c(3),c(4),c
(5),c(6),c(7)
3670 END DEFine
3680 DEFine PROCedure diagonal
3690    CLS #5: PRINT #5;' Position curs
or to other end of diagonal and press en
ter.'
3700    sec_point: cross: LINE x,y TO xo
ld,yold: d_mode=d_model
3710 END DEFine
3720 DEFine PROCedure buffer
3730    dirc(4)
3740    SELect ON dir_c
3750      =0: RETurn
3760      =1: LINE_R TO -12,0 TO 12,25
TO 12,-25 TO -12,0: y=y+25
3770      =2: LINE_R TO 0,12 TO 25,-12
TO -25,-12 TO 0,12: x=x+25
3780      =3: LINE_R TO 12,0 TO -12,-25
 TO -12,25 TO 12,0: y=y-25
3790      =4: LINE_R TO 0,12 TO -25,-12
 TO 25,-12 TO 0,12: x=x-25
3800    END SELect
3810 END DEFine
3820 DEFine PROCedure arrow
3830    dirc(4): FILL 1
3840    SELect ON dir_c
3850      =0: FILL 0: RETurn
3860      =1: LINE_R TO -4,0 TO 5,12 TO
 5,-12 TO -4,0
3870      =2: LINE_R TO 0,4 TO 12,-5 TO
 -12,-5 TO 0,4
3880      =3: LINE_R TO -4,0 TO 5,-12 T
O 5,12 TO -4,0
3890      =4: LINE_R TO 0,4 TO -12,-5 T
O 12,-5 TO 0,4
3900    END SELect
3910    FILL 0
3920 END DEFine
3930 DEFine PROCedure Edge_con
3940    dirc(4): FILL 1
3950    SELect ON dir_c
3960      =0: FILL 0: RETurn
3970      =1: LINE_R -1,21: plate_lr(i)
3980      =2: LINE_R 0,1: plate_ud(i)
3990      =3: LINE_R -1,0: plate_lr(i)
4000      =4: LINE_R -21,1: plate_ud(i)
4010    END SELect
4020    FILL 0
4030 END DEFine
4040 DEFine PROCedure trans
4050    CLS#5: PRINT #5;' Direction is t
aken by looking from the base      tow
ards the other terminals.': PAUSE 250
4060    dirc(4): IF dir_c=0: RETurn
4070    qu$=' Is the transistor to be NP
N ?'
4080    type=answer(qu$): IF type=-1:RETu
rn
4090    qu$=' Is the emitter either the
top- or left-most       connection ?'
4100    emit=answer(qu$): IF emit=-1: RE
Turn
4110    PRINT #5;' Move cursor to the de
sired position for the base   and press
enter.'
4120    cross: sec_point: cross
4130    SELect ON dir_c
4140      =1: LINE_R -15,0 TO 30,0; 0,2
0 TO -10,-20; -10,0 TO -10,20
4150        IF type
4160          IF emit: arr 4: ELSE L
INE_R 30,0: arr 1
4170        ELSE
4180          IF emit: LINE_R 0,-18:
 arr 3: ELSE LINE_R 20,-18: arr 2
4190        END IF
4200      =2: LINE_R 0,15 TO 0,-30; 20,
0 TO -20,10; 0,10 TO 20,10
4210        IF type
4220          IF emit: arr 5: ELSE L
INE_R 0,-30: arr 7
4230        ELSE
4240          IF emit: LINE_R -18,-1
0:arr 6: ELSE LINE_R -18,-20:arr 8
4250        END IF
4260      =3: LINE_R -15,0 TO 30,0; 0,-
20 TO -10,20; -10,0 TO -10,-20
4270        IF type
4280          IF emit: arr 2: ELSE L
INE_R 30,0: arr 3
4290        ELSE
4300          IF emit: LINE_R 10,18:
 arr 1: ELSE LINE_R 20,18: arr 4
4310        END IF
4320      =4: LINE_R  0,15 TO 0,-30; -2
0,0 TO 20,10; 0,10 TO -20,10
4330        IF type
4340          IF emit: arr 8: ELSE L
INE_R 0,-30: arr 6
4350        ELSE
4360          IF emit: LINE_R 18,-10
: arr 7: ELSE LINE_R 18,-20: arr 5
4370        END IF
4380    END SELect
4390 END DEFine
4400 DEFine PROCedure arr (no)
4410    SELect ON no
4420      =1: LINE_R TO 0,-10; -8,4 TO
8,6
4430      =2: LINE_R TO 0,10; 8,-4 TO -
8,-6
4440      =3: LINE_R TO 0,10; -8,-4 TO
8,-6
4450      =4: LINE_R TO 0,-10; 8,4 TO -
8,6
4460      =5: LINE_R TO -10,0; 4,-8 TO
6,8
4470      =6: LINE_R TO 10,0; -4,8 TO -
6,-8
4480      =7: LINE_R TO -10,0; 4,8 TO 6
,-8
4490      =8: LINE_R TO 10,0; -4,-8 TO
-6,8
4500    END SELect
4510 END DEFine
4520 DEFine PROCedure prompts
4530    BORDER 1,7: BORDER #4,1,7: CLS #
4: BORDER #5,1,7: CLS #5
4540    INK #4,7: LINE #4,0,50 TO 1100,5
0: INK #4,4
4550    CURSOR #4,2,26: PRINT #4;'F2 - M
OVE'
4560    CURSOR #4,2,5: PRINT #4;'F1 - LI
NE'
4570    CURSOR #4,372,26: PRINT #4;'F4 -
 ERASE'
4580 END DEFine
4590 DEFine PROCedure printer
4600    BORDER 1,0: BORDER #4,1,0: CLS #
4
4610    cross: CALL mem
4620    prompts
4630 END DEFine
4640 DEFine PROCedure msave
4650    cross:CLS#5: INPUT #5;' Enter fi
le name: ';dev$: IF dev$='':RETurn
4660    PRINT #5;' Saving to '&dev$&'_ca
d': PAUSE 100: CLS #5
4670    SBYTES dev$&'_cad',131072,32768
4680 END DEFine
4690 DEFine PROCedure mload
4700    cross:CLS#5: INPUT #5;' Enter fi
le name: ';dev$: IF dev$='':RETurn
4710    LBYTES dev$&'_cad',131072
4720    d_mode=2: x=262: y=300: LINE x,y
4730 END DEFine
4740 DEFine PROCedure get_no
4750    REPeat in_no
4760      get_key
4770      SELect ON k_code
4780        =27,48 TO 57: RETurn
4790      END SELect
4800    END REPeat in_no
4810 END DEFine
```

'armed with a little programming skill, the user should be able to adapt the program in any way'.

the left or right directions and so these are the only available.

Having given the direction, many of the logic gate components ask for the number of inputs. This has been restricted to two or four, as a three input gate can be drawn using a two input gate with an extra input drawn directly opposite the output. The user then has the choice of which input the cursor position is at. The inputs are numbered from top to bottom, or left to right depending on the direction of the gate. Once entered the gate is drawn taking into account the cursor position's input number.

The components, electrolitic capacitors, op-amps and transistors, are a bit more complex. These ask some yes-or-no questions which have to be answered before the component is drawn. Note at any stage the command can be abandoned by pressing the ESC key. Those commands that cannot be abandoned using ESC can be abandoned by simply pressing Enter.

The extra commands given when in commands2 mode are mostly self explanatory. The colour command simply changes the ink colour and is included because some television sets produce better high resolution graphics when drawn in red, but EASEL produces the darkest colour for white ink when a screen dump is printed out. The solution: draw it in red and before printing, change the ink to white.

The diagonal command uses the current cursor position for one end of the line and asks for the cursor to be moved to the other end. When the cursor is in the correct position, pressing Enter will draw the line.

The rest of the extra commands simply do what their name suggests. The Text command enters text at the current cursor position, the centre of the cross being taken as the bottom left-hand corner of the text character. The Print, Save and Load commands perform the tasks on the drawing that you would expect. Note, when giving a file name to the Save and Load commands, the device name (eg mdv1_etc.) must also be included. All filenames will also have the letters '_cad' appended to them automatically.

For those interested in improving or adding a section to the program a short description is necessary.

Lines 100-150 set up the variables and graphic and text windows; window #5 is used for the menu and window #1 for the drawing. Lines 160-620 then form the main program loop which tests for a key press and if necessary calls a procedure. The function keys and cursor keys are checked first, followed by those for menu 1 and then menu 2. The remainder of the program is made up of procedures which either perform a specific drawing routine or are called by several routines in an attempt to simplify the program.

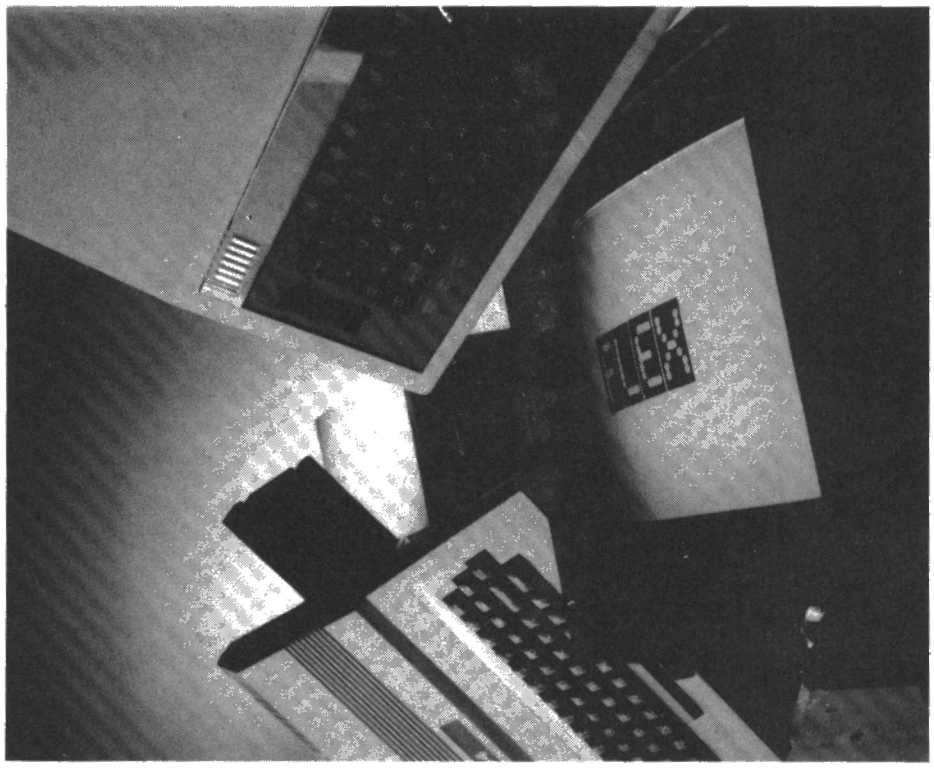
Many of the drawing routines have been simplified by using a direction flag. This allows the same routine to draw the component in two directions by using the direction flag to make the relevant co-ordinates negative.

For those who wish to add their own routines, it should be noted that there are routines to answer yes-or-no questions, routines to fetch key codes of capital letters independant of the computer's mode, and also routines to fetch the required direction, already written and available. Extra commands can simply be added to menu 2 and the test for the key press added in the second SELECT statement at line 460. This in turn can call the new procedure to execute the new command.

Armed with this knowledge and a little bit of programming skill, the user should be able to adapt the program in any way he or she needs. The limits of your imagination need be all that holds you back (and possibly a little bit of computing power) from that highly sophisticated, professional CAD system used in industry!

A DRAGON IN THE TUBE

Huw Jones' 6809 second processor for the BBC looks suspiciously like a Dragon 64. It should do because it is. This month he explains the software interface between the two machines.



The software controlling operation of the second processor comes in two sections. The programs are written in assembly language, one in 6502 code, the other in 6809. The main task of the supporting software is to enable the BBC micro to maintain constant communication with the 6809 processor and to interpret its command requests into I/O tasks. A simplified flow diagram of the program is shown in **Figure 1.**

Initialisation entails selecting the correct screen mode and colour (yellow and black), flushing all buffers, enabling the console for input and configuring the 8255 via its control register at $FEE3. At this point the BBC 'escape' key is disabled. Finally, a prompt is displayed on the BBC's screen and the computer waits for the user to press a key before it proceeds into the main loop proper. This pause allows the Dragon to be turned on at the correct time.

'The main task of the supporting software is to enable the BBC to maintain constant communication with the 6809 processor'.

The software continually scans the selected input stream, normally the keyboard, and any characters found are dispatched to the 6809. FLEX will automatically echo-back any character through the TUBE, at which point the BBC micro will pick it up and display it on the screen or send it to the currently selected output stream. This sequence of events applies to all ASCII characters in the range 0-$7F; codes above this are reserved for second processor system calls. When the 6809 sends such a call, the 6502 software will execute the associated command, listed in **Table 1.**

When a command commences, 6502BUSY goes active low, and the 6502 only returns it high when the execution phase is complete. With some of the commands, additional parameters must accompany the code byte in the order stated. Similarly, a number of commands return a variable number of values to the 6809 and/or an indication of successful completion via the message port. A 'good' finish sets $F9 as the message byte, otherwise the message port is cleared.

The assembled object code occupies approximately 3K and is booted from disk by typing:

TABLE 1. BBC command codes for FLEX.

Command	Parameters	Function
80		CURSOR EDIT
81		CURSOR EDIT
82		CURSOR EDIT
83		CURSOR EDIT
84		CURSOR EDIT
85		CURSOR EDIT
86		CURSOR EDIT
87		CURSOR EDIT
88		CURSOR EDIT
89		CURSOR EDIT
8A		CURSOR EDIT
8B		CURSOR EDIT
8C		CURSOR EDIT
8D		CURSOR EDIT
8E		CURSOR EDIT
8F		CURSOR EDIT
90	TRACK,SECTOR,DATA(256)>	read disk sector
91	TRACK,SECTOR,DATA(256)n	write disk sector
92	DRIVE NUMBER	select drive
93		init drive
94		enable printer o/p
95		disable printer
96		enable vdu
97		disable screen
98		enable RS423 o/p
99		disable RS423 o/p
9A	CHANNEL,SOUND GEN LIST	generate sound
9B	VDU CODE,VDU LIST	vdu driver
9C	CHANNEL,A/D> (MSB)	read a/d value
9D	STATUS> (NZ=CHAR READY)	check keyboard status
9E	STATUS> (NZ=CHAR READY)	get RS423 i/p status
9F	STATUS> (NZ=BUF EMPTY)	check RS423 o/p status
A0	OFFSET,BYTE)	read one byte from FRED
A1	OFFSET,BYTE<	write one byte to FRED
A2	OFFSET,BYTE<	read one byte from JIM
A3	OFFSET,BYTE>	write one byte to JIM
A4	OFFSET,BYTE<	read one byte SHEILA
A5	OFFSET,BYTE>	write one byte SHEILA
A6		abort FLEX (not used)
A7		MSGDIR=6809 to BBC
A8		MSGDIR=BBC to 6809
A9	DISK ERROR CODE>	request disk error info
AA	WP STATUS> ($B=WR PROT)	request disk WP status
AB	ADDR(H,L),CNT(H,L),DATA	transfer data to BBC memory
AC		enable RS423 i/p
AD		disable RS423 i/p
AE	BAUD RATE CODE	set RS423 baud rate (I/O)
AF	NO. OF TRACKS	format blank disk
B0	A REG,X REG,Y REG	perform OSBYTE call
B1		user defined function

NOTES:

1. > direction of byte is BBC to 6809
2. < direction of byte is 6809 to BBC
3. Unless shown, parameter byte direction is '<'
4. Command $AB expects 'CNT' bytes from 6809
5. Unless shown, parameters as per BBC MOS *FX calls

*GOFLEX

which loads and starts the software at $1E00. A set of jump tables located at $1E03 to $1E35 are reserved for command codes $80-$8F and $B1. These are currently not used and are reserved for cursor control and a user defined command respectively. Any specific I/O function required can be added to the 6502 code by patching in an extended jump at the relevant table entry and appending the code to the main program. Any code added in this way must end with an RTS instruction.

There are four tasks which the 6809 software must perform:

- Set up 64K map type 1 mode.
- Boot the FLEX system file from disk.
- Initialise the FLEX parameters for the second processor system and start the FLEX kernel.
- Handle all I/O and disk access calls from FLEX

On power-up, map type 0 is selected and the boot software appears in EPROM at $C000. If locations $C000 and $C001 contain the values $44, $4B (ASCII 'DK' – someone's initials at Microsoft?) then the software will auto-start, otherwise the boot procedure can be started by typing:

EXEC &HC002

The program first copies the contents of the BASIC ROMs and itself into the lower 32K of RAM before transfering control to the image of itself that it has just made. From this safe place, the SAM in the Dragon is switched to map type 1 which 'tacks-on' another block of 32K RAM at $8000-$FEFF. The program image now procedes to copy itself to $F800 onwards and the BASIC copy is placed back at $8000 from whence it originally came. Program control now transfers to the second image of the boot software and the system is ready to proceed to load FLEX. If this part of the procedure sounds complicated, imagine the problems of debugging the software across the map switching during the development of the project.

Conventionally, a FLEX system boots itself from a short disk-resident routine. This implies that a FLEX system disk is usually hardware specific. To circumvent this obstacle, the complete boot procedure is contained within the boot EPROM and the system should (in theory) be able to use any linked system disk.

Briefly, the boot program loads in the first sector on the system disk and checks that it has been 'linked' to the start of FLEX (a file named FLEX.SYS). If so, it locates this file and retrieves it sector by sector via the TUBE and places the code in the allotted space, $C000-$DFFF. When done, the disk driver and I/O jump tables within FLEX are modified to point to the routines resident within the boot software and then

TABLE 2. 6809 boot software TUBE vectors

Location	Function
FEF0	reserved for future use
FEF2	initialise tube interface
FEF4	check TUBE status (NZ if busy)
FEF6	set MSGDIR 6809 to BBC
FEF8	set MSGDIR BBC to 6809
FEFA	load control byte from TUBE
FEFC	check command completed (Z=OK)
FEFE	issue BBC command (code in A)

How to get your second processor kit and software.

A complete kit of parts for the TUBE interface, together with detailed information regarding the construction of the project is available from the Logic Shop at the address shown at the end of the article. The system software will be made available from Compusense.

After many weeks of hard use, the system has proved very reliable in operation and pleasant to work with. The BBC acts as an ultra fast I/O so that screenfuls of text appear in next to no time. Disk accesses take marginally longer than if they were implemented by an FDC section on the 6809 side, but it is disk searches that consume by far the bulk of waiting time. Experiments with sector interleaving to improve the throughput of the system have proved inconclusive. Although FLEX is workable with a single disk drive – just – the system really comes into its own with two, or more. If double sided drives are installed then Acorn's drive numbering convention effectively implements a four drive system which is very nice indeed!

Communication with laboratory equipment such as EPROM blowers and emulators is possible through the BBC's RS423 port. A download function code $AB has been built into the 6502 to expedite any data transfer required from the second processor.

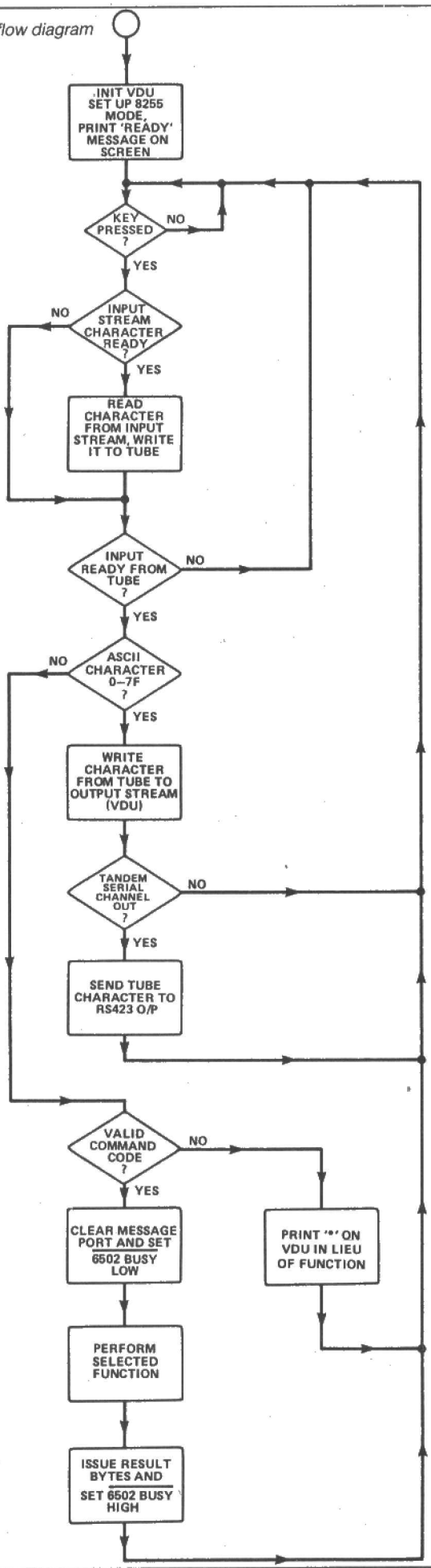
It is intended to develop a variety of peripherals for the system which will be driven by FLEX through the BBC's 1MHz bus.

The Logic Shop
Dunraven Place, Bridgent, Mid Glamorgan.
Telephone 0656 2656.
Will supply a complete kit of parts for the TUBE interface. Phone for details of price and availability.

Compusense
PO Box 169, 286D Green Lanes, London, N13 5XA.
Telephone 01 882 0681.
Will supply the system software. Phone for details of price and availability.

Compusense can also supply Dragon 64 computers.

Figure 1. Simplified flow diagram of the software.



suitable FLEX parameters are set for the BBC's keyboard. Control is then passed to FLEX.

All FLEX hardware calls are now forced through the boot software routines which activate the various TUBE requests. So that any FLEX utility may have access to the routines that service the TUBE (besides the standard FLEX I/O calls) a set of indirection operators is supplied at $FEF0-$FEFF and explained in **Table 2.** These greatly simplify the task of including any dedicated second processor commands within a FLEX utility and will remain fixed.

In order to fully integrate this new FLEX system, several new utilities must be prepared. A disk formatter is required to initialise a blank disk for use within the second processor arrangement. To keep things as simple as possible, the 'raw' format procedure, that is the inscribing of the ID and data fields on the disk by the 8217, is performed as a TUBE command in the BBC software, code byte $AF. The remaining operations to be carried out are accomplished by a custom FLEX utility called BBCFORM which supercedes the standard NEWDISK utility. This includes generating the directory, system record and free sector chain which is the very cornerstone of FLEX. To accommodate any type of disk drive, the utility first prompts the user to enter various parameters before entering the BBC format primitive.

All hard copy is obtained from the BBC printer port. Command codes $94 and $95 are used to select and terminate output to the MOS printer buffer. A new line printer driver has been prepared for the second processor which incorporates a custom version of the PRINT.SYS file required by FLEX XBASIC. The new command LPR replaces the original FLEX P utility.

Due to the vagaries in the map type 1 address decoding, the 6809 vectors are still obtained from the top of one of the BASIC ROMs. These vectors point to service routine jumps at $100-$111, which FLEX expects to be available for general purpose use. If FLEX writes 'garbage' prior to an interrupt occuring then spectacular crashes can ensue. In practice, the situation is nothing like as nasty as it appears. Interrupts are permanently disabled in the second processor (they are not required and printer spooling is not supported).

In addition, slightly amended versions of the few oft-used FLEX utilities that could corrupt the stack have been prepared. These offer added security by regarding $600 as the bottom of memory, just above screen RAM. Furthermore, a FLEX utility called PRESVEC restores the service jumps to point to routines in the boot software. This utility should be called before or after any FLEX software that demands interrupts. It should be stressed that none of these special precautions have been found necessary when using prototypes of the system, though if using a FLEX Debug package, interrupts should be disabled before using the trace function by setting CC to $50.

AMSTRAD SOUND SYSTEMS

Peter Green puts the Amstrad's complicated sound chip into action and reverts to the days at the original digital recording method – punched paper tape – for a Hurdy-Gurdy sound.

"The SOUND feature of the CPC464 is one of the most complex extensions to BASIC and is introduced in chapter six." So says the official Amstrad handbook, as if to apologise for the 7 parameters of the SOUND command – not to mention the 32 parameters of the ENV and ENT commands combined!

This article is about ignoring all those parameters and allowing the CPC464 to play away merrily all day in the style of the automatic barrel-organs and pianos. It's not quite a Hurdy-Gurdy or Melodeon, and it hasn't got a monkey attached to it, but it has got an endless stream of punched paper feeding it with fairground style music.

Machine sounds

All this is achieved by handling the computer's sound chip from machine code rather than from BASIC. Locomotive have made things as easy as possible by supplying a ROM routine which communicates in a straightforward manner with the IC involved, the AY-3-8912 (or "PSG") from General Instruments. This chip is not normally so easy to program (hence the complicated BASIC sound commands), but the Melodian program takes a fairly easy option by ignoring most of the complexities of the PSG's 15 internal registers. This article will show you how to squeeze standard electronic organ sounds from the PSG.

The tones produced by the low-cost variety of electronic organ are very straightforward: their frequency is rock-steady and the amplitude of sound is constant for the duration of each note. There is no envelope shaping, and the sound starts immediately and continues until it is either shut down or changed into a note of a different frequency. A non-micro example of such an organ (which Younger Readers may not have heard!) is the "Stylaphone", once widely advertised by no less a person than Rolf Harris in the days before he introduced cartoons on television. The PSG chip sounds much better than the transistor-driven Stylaphone, and with the power of the micro behind it, can play complicated pieces of music.

The PSG chip is "played" by a regular sequence of WRITEs to its various registers. The ROM routine MC SOUND REGISTER at #BD34 does the writing (which is somewhat involved due the complications of the CPC464 hardware), and the code written by the programmer consists of just three lines:

```
LD A,#07        ;register number 7
LD C,#38        ;the value to be written
CALL #BD34      ;perform the write
```

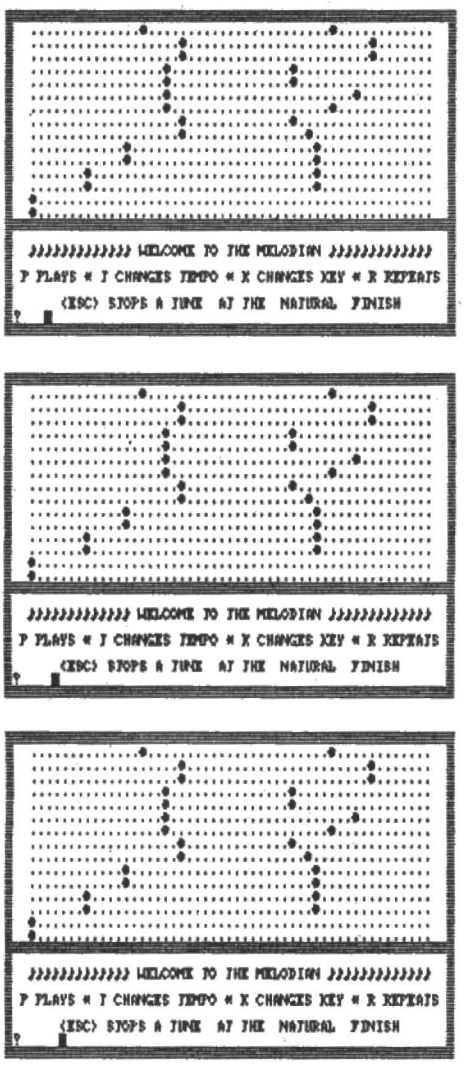
#38 is the code required by register 7, the control register, to set the simple tones required by the program. Setting register 7 is part of the initialisation procedure, as is setting the maximum volume (value 15) to the channel amplitude registers, 8, 9 and 10. At this point the PSG would begin to sound the tones which it held in its sound-period registers (registers 0 to 5) and to prevent this happening these registers are all loaded with the value zero.

Like the fairground organ with its punched music sheets freshly loaded, the PSG is now ready to play – it just needs some notes to be given to it! These are provided in the form of a large data table running from #5DCA to #6105 inclusive – the equivalent, not of the Melodean's sheet music, but of 46 musical bars. The MELODIAN shuffles these bars around and uses 16 of them chosen at random to produce its tune. More shuffling produces the next tune, and so on (and on and on . . .)

'The sound feature of the CPC464 is one of the most complex extensions to BASIC – so says the official Amstrad handbook'.

Automatic sequencing

The Melodian program runs from #6215 to #6394 and performs three separate tasks. It produces the random numbers needed to shuffle the bars around, then it translates the notes held in the musical bar table into bytes which the PSG can understand, and, finally, whilst it is playing each note it displays a representation of that note as a hole punched in an endlessly moving score. In the HISOFT assembler listing, the key sections are as follows:



Melodean screen dumps.

Lines 90-160: Lookup and note translation tables.

Lines 170-610: Initialisation of the playing order.

Specifically, lines 420-490 produce a random number in the range 0-3 which, when added to HL in line 560 cause 16 musical bars to be assembled in a holding area called CURST (line 110).

Lines 630-850: Initialise the PSG chip.

Lines 900-2140: a series of nested loops which play a sequence of 32 bars. The information in CURST is used twice, as each group of eight bars repeats. Embedded in the loops are the screen-handling routines. The paper is prepared by lines 1180-1230, and the notes punched by lines 1360-1430. The score is printed by lines 1710-1790. The score moves up the screen by the automatic scrolling action of a CPC464 window, so no code is required to produce the animation.

Within the music loop, there are two CALLs to the MC SOUND REGISTER routine for every note which needs to be played. This is because the tone-period register is a double register with a total length of 12 bits. After a note has been taken from the bar table, it is translated into a double-byte period value. The note value is in the Z80 B register (line 1340), and is used to point DE into a table of period values (lines 1490-1520). The two values found in the table are sent to the PSG by lines 1530-1570.

The BASIC

The BASIC program is responsible for a number of house-keeping tasks. The most important (not to say essential) task allocated to it is the loading of the two period-value tables. The period values, extracted from pages 1-3 in appendix VII of the Amstrad manual are found in BASIC lines 510-560. After the user has chosen the musical key in which the tunes are played, the data is transferred to the KEY1 and KEYTT2 areas of machine code at #6106-#61D1. A second task is to allocate one screen window for the animation and one for the screen menu. Having loaded the KEY tables and set a WINDOW 51 characters wide, the machine code will run quite happily on its own and continuous music can be achieved by a one line program:

```
1  CALL &6245:GOTO 1
```

The remaining BASIC program is present to make the program "user friendly" and allow it to run unsupervised.

Loading the program

The top of memory should be brought down to &5C00 using the MEMORY com-

LISTING 1. Melodean.

```
 4 MEMORY &5C00:LOAD "MC3"
 5 N=1:F=10:GOSUB 440:N=2:F=12:GOSUB 440:POKE &625C,1
 6 MODE 2:CLS:PRINT "DEMO MODE -- PRESS <P> TWICE, THEN <ENTER>"
 7 IF INKEY$="" THEN GOTO 7
 8 :
 9 :
10 MODE 2:CLS
20 W=207:U=237:RESTORE 30
30 DATA 1,17,25
40 FOR N=1 TO 3:READ X
50 LOCATE 13,X:PRINT STRING$(55,CHR$(W)):NEXT N
60 FOR N=1 TO 25:LOCATE 12,N:PRINT CHR$(W)
70 LOCATE 68,N:PRINT CHR$(W):NEXT N
90 WINDOW 13,67,18,24:CLS:LOCATE 3,2
100 PRINT STRING$(13,CHR$(237));" WELCOME TO THE MELODIAN ";
110 PRINT STRING$(13,CHR$(237)):PRINT
120 PRINT " P PLAYS * T CHANGES TEMPO * K CHANGES KEY * R REPEATS"
130 PRINT:PRINT "      <ESC> STOPS A TUNE  AT THE  NATURAL  FINISH"
140 INPUT N$
150 IF N$="P" OR N$="p" THEN GOTO 240
160 IF N$="T" OR N$="t" THEN GOTO 200
170 IF N$="K" OR N$="k" THEN GOTO 330
180 IF N$="R" OR N$="r" THEN GOSUB 290:GOTO 90
190 GOTO 270
200 PRINT "THE TEMPO IS ";PEEK(&625C)
210 PRINT: PRINT "SELECT TEMPO 1-255":INPUT N
220 IF N<1 OR N>255 THEN GOTO 200
230 POKE &625C,N:GOTO 90
240 WINDOW 15,65,2,16:LOCATE 1,17:K=0
245 :
250 REM:K=K+1:PRINT:PRINT STRING$(51,CHR$(U)):PRINT "TUNE NUMBER";K
260 REM:PRINT:PRINT STRING$(51,CHR$(U))
265 CALL &6245:GOTO 245
270 WINDOW 15,65,2,16:LOCATE 1,17:K=0
280 GOSUB 570:GOTO 280
290 WINDOW 15,65,2,16:LOCATE 1,17:Y=&6264
300 FOR X=&61D2 TO &61E1
310 POKE Y,PEEK(X):Y=Y+1:NEXT X
320 PRINT:PRINT "REPEATING":PRINT:CALL &6247:RETURN
330 CLS:PRINT "ALTER THE NOTE TABLE"
340 PRINT "TABLE 1 OR 2 ";:INPUT N
350 PRINT "THE KEY VALUE IS";
360 IF N=1 THEN PRINT PEEK(&6106)+PEEK(&6107)*256:GOTO 380
370 PRINT PEEK(&616C)+PEEK(&616D)*256
380 PRINT "C =1 E =5 G#= 9 C =13 E =17 G#=21 C =25"
390 PRINT "C#=2 F =6 A =10 C#=14 F =18 A =22 C#=26"
400 PRINT "D =3 F#=7 A#=11 D =15 F#=19 A#=23 D =27"
410 PRINT "D#=4 G =8 B =12 D#=16 G =20 B =24 D#=28"
420 PRINT:INPUT "START 1-28";F:IF F>28 THEN GOTO 420
430 GOSUB 440:GOTO 90
440 RESTORE 510
450 FOR X=1 TO F:READ Z:NEXT X
460 IF N=1 THEN S=&6106 ELSE S=&616C
470 IF N=1 THEN E=&616B ELSE E=&61D1
480 FOR X=S TO E STEP 2:READ N
490 POKE X,N-(INT(N/256)*256):POKE X+1,INT(N/256):NEXT X
500 RETURN
510 DATA 0,1911,1804,1703,1607,1517,1432,1351,1276,1204,1136,1073
520 DATA 1012,956,902,851,804,758,716,676,638,602,568,536,506
530 DATA 478,451,426,402,379,358,338,319,301,284,268,253
540 DATA 239,225,213,201,190,179,169,159,150,142,134,127
550 DATA 119,113,106,100,95,89,84,80,75,71,67,63,60,56,53
560 DATA 50,47,45,42,40,38,36,34,32,30,28,27,25,24,22,21,20
570 N=1:F=22:GOSUB 440:N=2:F=24:GOSUB 440
580 FOR X=1 TO 2
590 K=K+1:PRINT:PRINT STRING$(51,CHR$(U)):PRINT "TUNE NUMBER";K
600 PRINT:PRINT STRING$(51,CHR$(U)):CALL &6245:NEXT X
610 N=1:F=10:GOSUB 440:N=2:F=12:GOSUB 440
620 FOR X=1 TO 2
630 K=K+1:PRINT:PRINT STRING$(51,CHR$(U)):PRINT "TUNE NUMBER";K
640 PRINT:PRINT STRING$(51,CHR$(U)):CALL &6245:NEXT X
650 RETURN
660 SAVE "MEL3":SAVE "MC3",B,&5DCA,1500,&5DC5:STOP
999 WINDOW 1,80,1,25:CLS:STOP
```

mand and the two blocks of machine code entered. In the Hex dumps, the byte after the colon is a checksum byte and it can be ignored. Save the code using SAVE "MCT",B,&5DCA,1500 but do not run it yet. To generate the KEY tables you must type in the BASIC lines 5 and 440; to 560 inclusive. Save the partly written BASIC as "MELT" and execute with RUN 5. The melody should play, but, in the absence of a screen window, the display will look somewhat scrambled. The tune will also be played at top speed, since whole-screen scrolling takes very little time and the machine code will not be delayed as much as it ought to be! ESCAPE halts the program at the end of a 32-bar refrain, and ESC again will bring you back to BASIC. The full BASIC program allows for changes of key and tempo. Save the complete collection by RUN 660. The program instructions contained in lines 1 to 9 are intended for auto-running tapes – the normal start is RUN 10. There is plenty of scope for rewriting the BASIC to give various permutations of automatic key and temp changing. Tempo is changed with POKE &625C,x and key changes are made by setting variable N and F and calling the subroutine 440. N has a value of 1 or 2 depending on which KEY table is to be modified, and F should lie in the range 1 to 28, this being the number of notes above "B" (in octave –3) of the lowest note you wish Melodian to play.

LISTING 2. Data Table.

```
5DCA  2A 00 12 2A 00 12 27 00 0F 27 00 0F 2C 00 14 2C 00 14  : 64
5DDC  22 14 0C 22 14 0C 1F 14 0C 20 14 0C 24 00 00 2C 00 00  : 53
5DEE  2C 11 0D 2C 11 0D 25 11 0D 25 11 0D 29 00 00 29 00 00  : 6C
5E00  2C 00 08 2C 00 0C 27 00 14 29 00 14 27 00 0C 29 00 0C  : 4C
5E12  2C 27 08 2C 27 08 2C 18 14 2C 00 12 00 14 11 00 00 0F  : 80
5E24  20 11 0D 20 11 0D 25 11 0D 25 11 0D 29 00 00 29 00 00  : 54
5E36  29 14 0D 25 14 0D 29 14 0D 2C 14 0D 31 00 00 2C 00 00  : 84
5E48  25 00 0D 25 00 0D 25 00 08 25 00 08 00 00 01 00 00 01  : C0
5E5A  29 25 14 29 25 14 27 24 14 27 24 14 00 00 08 00 00 08  : 92
5E6C  24 00 14 22 00 14 24 00 14 25 00 14 27 00 00 24 00 00  : 2A
5E7E  29 00 0D 25 00 0D 24 00 0F 22 00 0F 20 00 03 1F 00 03  : 11
5E90  25 1D 0D 25 00 00 25 1D 0D 25 00 00 25 1D 0D 25 00 00  : 5C
5EA2  25 14 0D 25 14 0D 20 14 0D 20 14 0D 1D 00 00 1D 00 00  : 48
5EB4  2C 0F 0C 30 0F 0C 33 0F 0C 30 0F 0C 2C 0C 0F 2C 0C 0F  : B9
5EC6  2E 13 0F 2E 13 0F 2B 13 0F 2B 13 0F 27 13 0D 27 13 0D  : C8
5ED8  25 14 11 20 14 11 25 14 11 29 14 11 20 00 00 25 00 00  : 6C
5EEA  20 11 0D 20 11 0D 25 11 0D 25 11 0D 29 14 0D 29 14 0D  : 96
5EFC  29 14 0D 25 14 0D 29 14 0D 29 14 0D 2C 11 0D 2C 11 0D  : B8
5F0E  2C 00 0C 2C 00 0C 30 00 0C 33 00 0C 27 06 00 27 00 00  : 3F
5F20  25 00 0D 29 00 0D 2C 00 0F 27 00 0F 22 00 03 2B 00 03  : 2C
5F32  29 00 0D 29 00 0D 25 00 0D 25 00 0D 20 00 00 20 00 00  : 10
5F44  2A 00 12 29 00 11 27 00 0F 29 00 11 2A 00 12 2C 00 14  : 62
5F56  1B 00 0F 1F 00 0F 22 00 0F 27 00 0F 2B 00 0D 2E 00 0D  : 32
5F68  29 25 0D 29 25 11 25 29 14 25 29 11 29 25 19 29 25 0D  : 3D
5F7A  2A 18 14 29 18 14 2A 18 14 27 18 14 25 00 00 24 00 00  : 9D
5F8C  2B 16 0D 27 16 0D 22 16 0D 2E 16 0D 2B 00 00 27 00 00  : 80
5F9E  24 00 14 27 00 14 2C 00 14 27 00 14 24 00 08 24 00 08  : 46
5FB0  29 14 0D 25 14 0D 20 14 0D 20 14 0D 29 0D 14 29 0D 14  : A6
5FC2  2C 08 08 2C 08 08 2C 18 14 2C 14 12 00 13 11 00 11 0F  : 66
5FD4  29 00 14 25 00 14 27 00 14 24 00 14 20 00 00 20 00 00  : 29
5FE6  22 13 0F 22 13 0F 27 13 0F 27 13 0F 2B 16 0D 2B 16 0D  : B6
5FF8  22 00 0D 29 00 0D 27 00 0F 2C 00 0F 2B 00 03 2E 00 03  : 35
600A  2C 0F 0C 30 0F 0C 2C 0F 0C 27 0F 0C 24 00 00 24 00 00  : 63
601C  25 11 0D 25 14 0D 20 11 0D 20 14 0D 29 11 0D 29 14 0D  : 99
602E  2B 27 0D 2B 27 0D 27 2B 00 27 2B 00 2B 27 0D 2B 27 0D  : 20
6040  25 11 0D 24 11 0D 25 11 0D 29 11 0D 20 00 00 25 00 00  : 54
6052  25 11 0D 24 11 0D 25 11 0D 29 11 0D 20 00 00 20 00 00  : 4F
6064  24 00 08 25 00 08 27 00 08 24 00 08 22 00 00 20 00 00  : F6
6076  2C 11 0D 2C 11 0D 2A 11 0D 29 11 0D 27 00 00 25 00 00  : 6F
6088  22 00 12 22 00 12 2A 00 12 27 00 12 22 00 14 24 00 14  : 4B
609A  25 14 11 24 14 11 25 14 11 20 14 11 1D 00 00 19 00 00  : 58
60AC  25 00 0D 25 00 0D 19 00 0D 19 00 0D 00 00 01 00 00 01  : B2
60BE  2C 0F 0C 2C 0F 0C 2C 0F 0C 27 0F 0C 30 00 00 30 00 00  : 77
60D0  29 14 0D 29 14 0D 25 14 0D 29 14 0D 2C 0D 11 31 0D 11  : BD
60E2  29 11 0D 29 14 0D 25 11 0D 25 14 0D 20 11 0D 20 14 0D  : 99
60F4  27 00 0D 27 00 0D 22 00 0D 22 00 0D 2B 00 00 2B 00 00  : 1C
```

A MEMOTECH METAMORPHOSIS

Richard Sargent reports on a stunningly clever piece of software which turns a Memotech into a Spectrum. We do not lie!

Take one Memotech MTX512 computer, a wedge-shaped MTX ROM-Pack, and a Spectrum Games Cassette – *Jetpac* will do nicely. Load Spectrum tape into Memotech . . . observe Sinclair's flashing border on screen . . . and play Jetpac using the usual keys and at the usual speed . . .

Science Fiction, perhaps? Freelance software designer Tony Brewer didn't think so, and to prove it he has the MTX512 currently running 20 Spectrum machine-code games. Memotech is as you might expect, rather pleased.

How it's done is more fascinating than why it's done. Memotech, very sensibly, see this as one more facility available to users of their machines; underlining the claim that the MTX range of computers and peripherals constitute powerful and versatile tools. But the transfer was made to prove that software compatibility *can* be achieved if the hardware involved is flexible enough.

Why it works

The Spectrum has a screen resolution of 256 x 192 pixels, which is the same as that of the MTX. Both computers run using Z80 code and both can cope with Z80 non-maskable interrupt. The MTX runs slightly faster – 4MHz as opposed to 3.5MHz – and, if there had to be a speed difference, this way round is the easier to cope with. So far so good, but what of the machines' operating systems? There is not a block of code in the Spectrum ROM which is anything remotely like that in the MTX ROM, and it would break all the copyright rules under the sun to put Sinclair's ROM into the MTX. The solution to this problem is simply to ignore the Spectrum ROM: there are very few routines in it which are of use to the writers of fast arcade games, and most Spectrum machine-code games make just one or two calls to the ROM. To put it another way, the code on the cassette tape is virtually self-contained, and a small amount of supervision by a friendly CPU will cause it to run.

How it works

Without giving away too many of Tony Brewer's ideas, it is possible to give a reasonably detailed account of how the system works. First, you need an MTX with 64K of RAM, so it has to be an MTX512 or an expanded MTX500. Then you will need to purchase the small box-of-tricks (price to be announced) which plugs into the Memotech ROM slot, and with that you will also receive the first of a series of cassette tapes which allow you to play *selected* Spectrum games. Memotech says the cassette tapes will be priced at less than the average games-tape, and each will allow between 0 and 12 specific Spectrum games to be used on the MTX. The first one, which is now ready to be produced in quantity, is a generous offering and contains the supervisory code to enable the twenty games listed in **Table 1** to be played.

TABLE 1. Spectrum games available

ARCADIA	ASTRONUT
ATIC ATAC	DECATHLON
GRIDRUNNER	HUMPTY DUMPTY
HUNCHBACK	JETPAC
JETSET WILLY	JUMP CHALLENGE
LASERWARP	POITY PIGEON
PROJECT FUTURE	SPECTIPEDE
STARION	STOP THE EXPRESS
TORNADO	TRAXX
TWIN KINGDOM VALLEY	
WORSE THINGS HAPPEN	

Once the tape is loaded into the MTX, an auto-run is performed and the MTX proceeds to turn itself (partially) into a Spectrum. The first indication that this has happened is the appearance of the on-screen menu, listing the Spectrum games that can now be loaded into the MTX. Naturally, this menu is in the Spectrum character set and uses Spectrum colours.

Internally, a more important change has taken place. The banked-memory which the 512 possesses moves from its normal position (PAGE 1, 8000H-BFFFH) to PAGE 0, 0000H-3FFFH, giving PAGE 0 a complete range of RAM from 0 to 64K. The Spectrum character shape-table is created at 3D00H-3FFFH, while 4000H-5CBSH is put aside for the "Spectrum screen" and the "Spectrum system variables". This leaves 5CB6H-FFFFH free to accept the Spectrum-game's code.

The supervisory code lies somewhere in the Z000H-5CFFH area. One major, purpose-written routine is the "Load Spectrum-format tape" (and there is also a "Save Spectrum-format tape" to cater for games where you can save a partly played version), but the main effort of coding is the routine which takes the display from 4000H (Spectrum display file) and 5800H (Spectrum colour attributes) and passes it to the 16K of video RAM used by the MTX's Video Processing chip (the VDP). This is where the artistry comes in. The task is performed using interrupts, but even so it takes two passes to move the relatively small Spectrum video RAM (size 1B00H) to the larger VDP RAM (size 4000H); this does not reduce the speed of the game, but does cause the graphics to move less smoothly: a point which is noticeable only when large sprite graphics are involved. For all other games there is no visual difference between the Spectrum version and the Memotech version.

'Fooling the code is the main task of the hardware, which although in a ROM-pack doesn't contain a ROM, but two custom chips'.

No copied code

Apart from the legal implications of copying Spectrum code, the challenge of not doing so appealed to Tony Brewer. Thus when a games program uses a call to the Spectrum ROM, something has to be there at the "ROM address" to intercept the flow of the program. An example is the CLS routine, widely used by games-programmers as a quick way of cleaning out the area of RAM from 4000H-5B00H. It's at 0D6BH, so the MTX has screen-clearing code at 0D6BH too. Similar trapping has to be done for the often used Z80 RST addresses (print-to-screen at 0010H is often used), and for the interrupt RST at 0038H.

BEEPER (03B5H) is rather more difficult. Sound is not used on the Memotech version of Spectrum games, since the effect doesn't warrant the high cost of implementing it. Nevertheless, the call is intercepted and the games code thinks the beeper-port exists: this is necessary to avoid program crashes.

Fooling the code is the main task of the hardware, which, although in a ROM-pack case, doesn't contain a ROM. What it does contain is five chips, two of which are custom-blown PALs. The PAL (Phase Alternation by Line) is the cheap younger brother of the ULA, and is used extensively in decoding circuits, and (as a side benefit) to prevent inquisitive constructors working out how a circuit operates. The other three chips are standard devices. The other duty of the hardware is to pretend to be a Spectrum keyboard: with some help from MTX code and MTX interupts, the keypresses (on the Memotech keyboard) are translated into Spectrum-style keypresses and joystick movements.

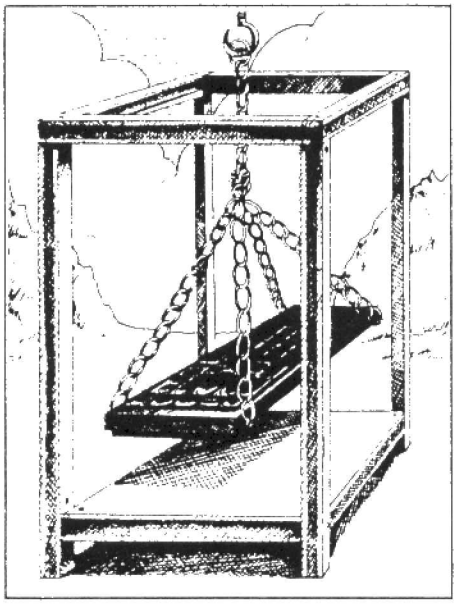
It is doubtful whether the whole package (tentatively called the Spectrum Speculator) will sell in any quantity now that the games-market has died down, but full marks to Memotech (and Tony Brewer) for doing it anyway.

QSHELL

More command routines for Adam Denning's user-friendly QDOS front end.

QSHELL UPDATE

QSHELL is still unfinished. However, the full version will be completed for the first issue of *Computing Age* (see pages 14 and 15). Adam Denning will then provide a comprehensive operator's guide and we will give details of how to obtain your own copy of QSHELL on Microdrive.



This month we look at some more of the command routines. The best way to do this is to go down the listing looking at each routine in turn, describing both the command and the code which implements it.

CLS_RTN handles the CLS command. All this does is clear the window attached to channel 1 by jumping to the routine which set it up in the first place – **WINDOW1.**

HELP_RTN deals with the HELP command. For this to work, there must be a file called **'HELP_TXT'** on the default drive; it doesn't matter what you put into this file, but as it is supposed to help the user a brief summary of each command is probably the most useful content. The routine works by fooling the command line interpreter into thinking that the line **'MORE HELP_TXT'** has been typed by the user. This causes the MORE routine to be invoked on the help_txt file, displaying it page by page. As an aside, it also shows the basic technique which will be used to implement batch processing commands later on.

WNDW_RTN is called by the WINDOWS command. It is similar to CLS, but clears and re-draws all three windows.

MED_RTN and **DIR_RTN** are combined, as they perform essentially the same function. MED or MEDIUM invokes MED_RTN, and DIR invokes DIR_RTN. MED interrogates the disk (or microdrive) to find its volume name and number of free sectors, while DIR produces a full directory as well as the MEDIUM information.

If there is no command tail after 'DIR', the routine uses the default device, otherwise it attempts to open the specified device to catalogue it. For this to work, the device specified must be a 'directory device', which means that it has a directory device driver rather than a normal device driver. If the OPEN_DIR call fails, a message is printed out and an error generated in D0.

Successful opening results in SHOW_DIR, where the volume and device names are printed out. The volume name is found with the FS_MDINF routine, which also returns the total number of sectors and the number of free sectors in D1. If DIR is being executed rather than MED (shown by DZ,B being non-zero), the files on the device must be listed to channel 1. This is done at DIR_LOOP.

The routine CHK_ESC is called to see if the ESC key has been pressed. If it has, the directory listing stops straight away and jumps to the section which prints out the sector details. Otherwise, the current filename is printed out. If this filename proves to be of zero length, there is still a file there – file length is non-zero – so the filename is printed as 'Un-named file'. SD_TAB is then called to move the cursor to column 40, and the length of the file in bytes is printed out, in decimal, using the DICITC routine. CD_TAB is called again to tab to column 55, and CN_DATE / UT_MTEXT are used to print out the update date saved in the file's header. This will be random for microdrives, but will be valid for disks. The loop is repeated until EOF is met, or ESC is pressed.

At DIR_END, the two halves of D1,L are printed out between relevant messages to show the number of sectors free and the total number of sectors on the medium. Notice that we use the QDOS routine UT_MINT to convert the numbers to ASCII. Once this has finished, a final line feed is printed out and the directory device is closed.

DIGITS is a standard binary to decimal ASCII conversion routine, but treats the number in D1 as unsigned and uses progressive subtraction rather than long division. Mainly because it's far easier to write! Notice that leading zeros and leading spaces are suppressed.

The next routine is **TYPE_RTN,** which is called whenever TYPE is invoked. TYPE simply copies a named file to channel 1, but can be stopped at any time by pressing ESC. It starts by checking that a filename

has been entered as well as the keyword, as it obviously needs a file to type! It then tries to open this, reporting requisite errors if it fails. If successful, a loop is entered in which sector-sized blocks (512 bytes) are read from the file and copied to the console. If any error other than EOF occurs during this loop, the command is aborted and an error message printed. When TYPE has finished, it closes the file and returns to the CLI.

QUIT_RTN is the code to deal with the QUIT command. It prints out a message telling the user to press 'Y' (or 'y') if he really wants to leave QSHELL, and then it kills the job if the answer is 'yes'. As the CLI was invoked from BASIC, killing it returns control to the SuperBasic interpreter. As the channels used by QSHELL are owned by BASIC, they are not closed.

ATOI is a utility routine used by numerous command routines to convert ASCII strings into decimal numbers. It is industry standard.

The next group of routines all perform various actions upon jobs running in the machine, and all are capable of identifying a job from its standard format name or its job number. **KIL_RTN** is called by KILL, and kills the specified job and all its children. **SUS_RTN** is called by SUSPEND, and suspends the specified job indefinitely (ie until it is killed or released with RELEASE).

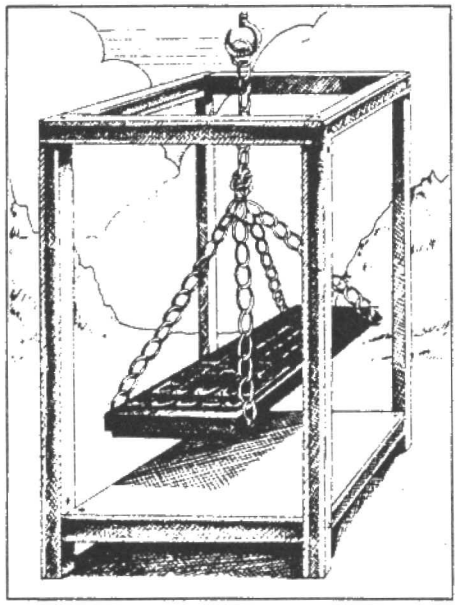
REL_RTN is called by RELEASE, and releases a suspended job. **PRI_RTN** is called by PRIORITY, and alters the priority of the specified job to the given new value. This routine uses ATOI to convert the new priority entered into a number, and then checks the byte value of D1 to ensure that it is less than 128 (by checking the sign flag after a TST).

STT_RTN is called by START, and activates a LOADed job, passing channel IDs and parameter strings as required. If the job is already active, a message is printed out and the command aborts with error. Otherwise, it follows the same code as

'DIR-END shows the total number of sectors free'.

RUN but does not load or create the job (as it already exists!).

The routine which does all the hard work of converting a job name or number into a real job IS is **JOB_GET.** This first checks to see if the first character of the command tail is a digit. If it is, it assumes that the



parameter entered is a number rather than a name, and goes off to find the job by number. If ATOI fails, or if the job parameter is not a number, it attempts to find the job by name, which is far more complicated. It has to scan the job tree using MY_JINF, checking the word at offset 6 of each job to ensure that it is in standard format (as only standard format jobs have names), and then comparing the string at offset 8 with the string entered. String comparison type 1 is used, so that the case of the letters in each string is not significant. If the job is not found, an error message is printed out and the routine returns one level higher by adding 4 to A7 before executing an RTS.

The final routine shown this month is the MORE command routine, **MRE_RTN.** This is fairly similar to TYPE, except that it prints out a screenful at a time, pausing after each one to display 'MORE?'. If the user presses ESC, 'n' or 'N', the routine finishes, otherwise the next screenful is shown.

It's fairly awkward to work out when a screen has been filled, as you have to interrogate the channel to find its width and its height. Subtractions and divisions then indicate how many screen lines have been filled so far.

It works like this: after opening the specified file, SD_CHENQ is called to find the dimensions of channel 1; although the window has already been set up, and therefore the dimensions are known, it's best to ask QDOS once more in case the window size is to be altered later; it also gives the routine a general purpose application.

The screen is cleared with SD_CLEAR between pages, and a line of up to 256 characters is read in from the file with IO_FLINE. The error return from this is saved in D4 for later examination. The length of the line is then divided by the screen width to find out how many screen lines the line just read-in will occupy, and then this is added to the record of the current cursor position in the Y direction. IO_SSTRG then prints the text out to the screen, and subtractions are performed on the screen length and current Y position to see if the screen is full. If it isn't, and if no error occurs in IO_FLINE, the routine jumps to do it all over again.

Otherwise, the 'More?' message is printed and the cursor enabled. A keypress is collected and compared with 'N', 'n' and ESC. If it matches any of these, the routine finishes, otherwise it goes off to do the next page.

QSHELL Command routines

```
* Routine to clear channel 1

CLS_RTN  BRA      WINDOW1

* A routine to provide help

HELP_RTN LEA.L    HLP_MES,A2
         LEA.L    D_SPACE,A4
         BSR      STRING_MV
         MOVEA.L  A2,A1
         MOVE.W   (A1)+,D1
         ADDQ.L   #4,A7
         BRA      REPT_CMD

HLP_MES  DC.W     13
         DC.B     'MORE HELP_TXT',0

* Routine to reset all windows

WNDW_RTN BSR      WINDOW0
         BSR      WINDOW1
         BSR      WINDOW2
         MOVE.L   DEFAULT,A2
         BRA      PUT_DEFT

* Device status routine

MED_RTN  MOVEQ    #0,D7
         BRA.S    DIR_MED

* Device directory routine

DIR_RTN  MOVEQ    #1,D7
DIR_MED  MOVE.L   A1,D0
         BNE.S    DIR_TAIL
         MOVE.L   DEFAULT,A1
DIR_TAIL MOVEA.L  A1,A5
         MOVEA.L  A1,A0
         MOVEQ    #-1,D1
         MOVEQ    #OPEN_DIR,D3
         QDOS     IO_OPEN,2
         TST.L    D0
         BEQ.S    SHOW_DIR
         MOVE.L   D0,D4
         LEA.L    DIR_MES,A1
         MOVE.L   CHAN_0,A0
         VECTOR   UT_MTEXT,2,CALL
         MOVEA.L  A5,A1
         VECTOR   UT_MTEXT,2,CALL
         LEA.L    DIR_MES2,A1
         VECTOR   UT_MTEXT,2,CALL
         MOVE.L   D4,D0
         VECTOR   UT_ERRO,2,CALL
         MOVEQ    #1,D0
         RTS
SHOW_DIR MOVEA.L  A0,A4
         LEA.L    DIR_MES3,A1
         MOVE.L   CHAN_1,A0
         VECTOR   UT_MTEXT,2,CALL
         MOVEA.L  A5,A1
         VECTOR   UT_MTEXT,2,CALL
         LEA.L    DIR_MES4,A1
         VECTOR   UT_MTEXT,2,CALL
         MOVE.L   A4,A0
         MOVEQ    #-1,D3
         LEA.L    D_SPACE,A1
         QDOS     FS_MDINF,3
         MOVE.L   D1,-(A7)
         MOVEQ    #12,D2
         LEA.L    D_SPACE,A1
         MOVE.L   CHAN_1,A0
         MOVE.W   #$0A0A,IO(A1)
         QDOS     IO_SSTRG,3
         TST.B    D7
         BEQ      DIR_END
DIR_LOOP BSR      CHK_ESC
         BEQ.S    DIR_END
         MOVEQ    #hdrlen,D2
         LEA.L    D_SPACE,A1
         MOVEA.L  A4,A0
         QDOS     IO_FSTRG,3
         TST.L    D0
         BNE.S    DIR_END
         SUBA.L   D2,A1
         MOVE.L   (A1),D1
         BEQ.S    DIR_LOOP
         ADDA.L   #$E,A1
         MOVE.L   CHAN_1,A0
         MOVE.W   (A1)+,D2
         BNE.S    HAS_NAME
         LEA.L    DIR_MES5,A1
         VECTOR   UT_MTEXT,2,CALL
         BRA.S    DONE_NAME
HAS_NAME
         QDOS     IO_SSTRG,3
DONE_NAME MOVEQ   #40,D1
         MOVEQ    #-1,D3
         QDOS     SD_TAB,3
         MOVE.L   D_SPACE,D1
         SUBI.L   #hdrlen,D1
         LEA.L    D_SPACE,A1
         BSR.S    DIGITS
         VECTOR   UT_MTEXT,2,CALL
```

```
        MOVEQ   #55,D1
        MOVEQ   #-1,D3
        QDOS    SD_TAB,3
        MOVE.L  D_SPACE+$34,D1
        LEA.L   D_SPACE+$40,A1
        SUBA.L  A6,A6
        VECTOR  CN_DATE,2,CALL
        VECTOR  UT_MTEXT,2,CALL
        MOVEQ   #linefeed,D1
        MOVEQ   #-1,D3
        QDOS    IO_SBYTE,3
        BRA     DIR_LOOP
DIR_END MOVEQ   #0,D1
        MOVE.W  (A7),D1
        MOVE.L  CHAN_I,A0
        VECTOR  UT_MINT,2,CALL
        LEA.L   DIR_MES6,A1
        VECTOR  UT_MTEXT,2,CALL
        MOVE.L  (A7)+,D1
        VECTOR  UT_MINT,2,CALL
        MOVEQ   #linefeed,D1
        MOVEQ   #-1,D3
        QDOS    IO_SBYTE,3
        MOVEA.L A4,A0
        QDOS    IO_CLOSE,2
        MOVEQ   #0,D0
        RTS
* Converts long integer in D1 to ASCII
* at A1. Leading zeros/spaces
* suppressed. Must leave A1 unchanged
* Corrupts D0,D1,D4,D5,D6 and A2

DIGITS  MOVE.L  A1,-(A7)
        LEA.L   TENTAB,A2
        ADDQ.L  #2,A1
        MOVEQ   #0,D0
        MOVEQ   #0,D4
        MOVEQ   #0,D6
ONEDIG  MOVEQ   #0,D5
DIGONE  SUB.L   (A2),D1
        BCS.S   GOTDIG
        ADDQ.B  #1,D5
        BRA.S   DIGONE
GOTDIG  ADD.L   (A2)+,D1
        ADDQ.B  #1,D0
        TST.B   D5
        BNE.S   NOTLDZ
        TST.B   D4
        BNE.S   NOTLDZ
        CMPI.B  #10,D0
        BNE.S   NEXTDIG
NOTLDZ  ADDI.B  #'0',D5
        MOVEQ   #1,D4
        ADDQ.W  #1,D6
        MOVE.B  D5,(A1)+
NEXTDIG CMPI.B  #10,D0
        BNE.S   ONEDIG
        MOVE.L  (A7)+,A1
        MOVE.W  D6,(A1)
        RTS

TENTAB  DC.L    1000000000
        DC.L    100000000
        DC.L    10000000
        DC.L    1000000
        DC.L    100000
        DC.L    10000
        DC.L    1000
        DC.L    100
        DC.L    10
        DC.L    1

DIR_MES DC.W    25
        DC.B    'Cannot open direct'
        DC.B    'ory on ',0
DIR_MES2 DC.W   4
        DC.B    ' as '

DIR_MES3 DC.W   14
        DC.B    10,'Directory of '

DIR_MES4 DC.W   18
        DC.B    ':  Volume name is '

DIR_MES5 DC.W   13
        DC.B    'Un-named file',0

DIR_MES6 DC.W   21
        DC.B    '  sectors free out'
        DC.B    'of ',0

* The file typing routine

TYPE_RTN MOVE.L A1,D0
        BNE.S   TYPE_FILE
        LEA.L   TYP_MES,A1
        MOVE.L  CHAN_O,A0
        VECTOR  UT_MTEXT,2,CALL
        MOVEQ   #1,D0
        RTS
TYPE_FILE MOVEA.L A1,A0
        MOVEQ   #-1,D1
        MOVEQ   #OPEN_INS,D3
        BSR     OPEN_THIS
        TST.L   D0
        BEQ.S   TYPE_OPEN
        MOVEA.L A0,A4
        MOVE.L  D0,D4
        LEA.L   TYP_MES1,A1
        MOVE.L  CHAN_O,A0
        VECTOR  UT_MTEXT,2,CALL
        MOVEA.L A4,A1
        VECTOR  UT_MTEXT,2,CALL
ABRT_TYPE LEA.L DIR_MES2,A1
        VECTOR  UT_MTEXT,2,CALL
        MOVE.L  D4,D0
        VECTOR  UT_ERRO,2,CALL
        MOVEQ   #1,D0
        RTS
TYPE_OPEN MOVEA.L A0,A4
        LEA.L   D_SPACE,A5
TYPE_LOOP MOVEQ #-1,D3
        MOVEA.L A4,A0
        MOVEA.L A5,A1
        MOVE.W  #sector,D2
        QDOS    IO_FSTRG,3
        MOVE.L  D0,D4
        MOVE.W  D1,D2
        MOVEA.L CHAN_I,A0
        MOVEA.L A5,A1
        QDOS    IO_SSTRG,3
        BSR     CHK_ESC
        BEQ.S   ERR_OK
        TST.L   D4
        BEQ.S   TYPE_LOOP
TYPE_ERR CMP.W  #ERR_EF,D4
        BEQ.S   ERR_OK
        LEA.L   TYP_MES2,A1
        MOVE.L  CHAN_O,A0
        VECTOR  UT_MTEXT,2,CALL
        MOVE.L  D4,D0
        VECTOR  UT_ERRO,2,CALL
ERR_OK  MOVEA.L A4,A0
        QDOS    IO_CLOSE,2
        RTS
TYP_MES DC.W    43
        DC.B    '  Please use TYPE '
        DC.B    'like this: TYPE '
        DC.B    '<filename>',10,0

TYP_MES1 DC.W   17
        DC.B    'Cannot open '
        DC.B    'file ',0

TYP_MES2 DC.W   29
        DC.B    'Error occurred '
        DC.B    'during TYPE - ',0

QUIT_RTN LEA.L  QUT_MES,A1
        MOVE.L  CHAN_O,A0
        VECTOR  UT_MTEXT,2,CALL
        MOVEQ   #-1,D3
        QDOS    IO_FBYTE,3
        MOVE.B  D1,D2
        MOVEQ   #linefeed,D1
        QDOS    IO_SBYTE,3
        CMPI.B  #'Y',D2
        BEQ.S   YES_QUIT
        CMPI.B  #'y',D2
        BEQ.S   YES_QUIT
        RTS
YES_QUIT BRA    CS_EXIT
QUT_MES DC.W    28
        DC.B    'Press Y or y to '
        DC.B    'leave QSHELL'
* A routine to convert a string (SHELL
* format - terminated with 0) into a
* number. Returns with zero flag reset
if bad number, else value in D0
* Corrupts D0, D1 and D2

ATOI    MOVEQ   #'0',D2
        MOVEQ   #0,D0
        MOVEQ   #0,D1
DO_KNUM MOVE.B  (A0)+,D1
        BEQ.S   END_KNUM
        SUB.B   D2,D1
        BMI.S   BAD_NUMB
        CMPI.B  #9,D1
        BGT.S   BAD_NUMB
        MULU    #10,D0
        ADD.L   D1,D0
        BRA.S   DO_KNUM
END_KNUM MOVEQ  #0,D1
        RTS
BAD_NUMB MOVEQ  #1,D1
        RTS

* The routine to kill a job

KIL_RTN MOVE.L  A1,D0
        BNE.S   KIL_CMD
BACK_KILL LEA.L KIL_MES,A1
KILL_BACK MOVEA.L CHAN_O,A0
        VECTOR  UT_MTEXT,2,CALL
        MOVEQ   #1,D0
        RTS
KIL_CMD BSR     JOB_GET
        MOVEQ   #0,D3
        QDOS    MT_FRJOB,1
        RTS

* The routine to suspend a job

SUS_RTN MOVE.L  A1,D0
        BEQ.S   BACK_KILL
        BSR     JOB_GET
        SUBA.L  A1,A1
        MOVEQ   #-1,D3
        QDOS    MT_SUSJB,1
        RTS

* The routine to release a job

REL_RTN MOVE.L  A1,D0
        BEQ.S   BACK_KILL
        BSR     JOB_GET
        QDOS    MT_RELJB,1
        RTS
* The routine to alter the priority of
* a job

PRI_RTN MOVE.L  A1,D0
        BEQ.S   BACK_KILL
        MOVEQ   #1,D1
        MOVEQ   #' ',D2
        BSR     GET_ARGN
        MOVEA.L A1,A4
        MOVEA.L A0,A1
        BSR     JOB_GET
        MOVE.L  D1,-(A7)
        MOVEA.L A4,A1
        BSR     NEW_CMDS
        MOVE.L  A1,D0
        BEQ.S   BAD_PRIO
        BSR     NEW_CMDS
        BSR     ATOI
        BNE.S   BAD_PRIO
        TST.B   D0
        BMI.S   BAD_PVAL
        MOVE.L  D0,D2
        MOVE.L  (A7)+,D1
        QDOS    MT_PRIOR,1
        RTS
BAD_PRIO LEA.L  PRI_MES,A1
PRIO_RET MOVEA.L CHAN_O,A0
        VECTOR  UT_MTEXT,2,CALL
        MOVEQ   #1,D0
        ADDQ.L  #4,A7
        RTS
BAD_PVAL LEA.L  PRI_MES1,A1
        BRA.S   PRIO_RET

PRI_MES DC.W    77
        DC.B    'Please use PRIORITY'
        DC.B    ' like this: '
        DC.B    'PRIORITY <job name>'
        DC.B    ' | <job number> '
        DC.B    '<priority>',10,0

PRI_MES1 DC.W   39
        DC.B    'The priority must '
        DC.B    'be between 0 and '
        DC.B    '127',10,0
* The routine to activate a previously
* loaded job
STT_RTN MOVE.L  A1,D0
        BEQ     BACK_KILL
        MOVEQ   #1,D1
        MOVEQ   #' ',D2
        BSR     GET_ARGN
        MOVEA.L A1,A4
        MOVEA.L A0,A1
        BSR     JOB_GET
        TST.B   D3
        BEQ.S   STT_SUSP
        LEA.L   STT_MES,A1
        BRA     KILL_BACK
STT_SUSP MOVEM.L A2-A4/D1,-(A7)
        MOVEQ   #0,D2
        QDOS    MT_JINF,1
        MOVE.L  A0,4(A7)
        MOVE.L  12(A7),A1
        MOVE.W  (A1),D1
        BSR     NEW_CMDS
        MOVE.L  A1,12(A7)
        MOVEA.L A0,A1
        BSR     NEW_PTR
        MOVE.L  A0,8(A7)
        ADDQ.L  #2,A0
        MOVEQ   #0,D7
STT_MOVE MOVE.B (A1)+,(A0)+
        BEQ.S   STT_MOVD
        ADDQ.W  #1,D7
        BRA.S   STT_MOVE
STT_MOVD MOVEA.L 8(A7),A0
        MOVE.W  D7,(A0)
        MOVEQ   #0,D7
        BRA     LD_OK_J

STT_MES DC.W    27
        DC.B    'This job is already'
        DC.B    ' active',10,0

KIL_MES DC.W    56
        DC.B    'You must specify a '
        DC.B    'job name or number '
        DC.B    'with this command'
        DC.B    10
* The routine to find a job whose name or
* job number is pointed to by A1. If the
* job is found, then D1 holds its ID on
* return, otherwise the routine does not
* return to the first caller, but goes a
* level higher

JOB_GET MOVE.B  2(A1),D0
        CMPI.B  #'0',D0
        BLT.S   JOB_NME
        CMPI.B  #'9',D0
        BGT.S   JOB_NME
        MOVE.L  A1,-(A7)
        MOVEQ   #3,D0
        BSR     NEW_HEAP
        MOVE.L  A1,-(A7)
        MOVEA.L 4(A7),A0
        MOVE.W  (A0)+,D0
        SUBQ.W  #1,D0
MV_JOBNM MOVE.B (A0)+,(A1)+
        DBRA    D0,MV_JOBNM
        CLR.B   (A1)
        MOVEA.L (A7)+,A0
        BSR     ATOI
```

```
        BNE.S   JOB_NAME
        ADDQ.L  #2,A7
        MOVE.W  D0,(A7)
        MOVEQ   #0,D1
JOB_NUM MOVEQ   #0,D2
        MOVE.L  D1,D4
        QDOS    MT_JINF,1
        TST.L   D0
        BNE.S   NO_JOBN
        CMP.W   (A7),D4
        BEQ.S   GOT_JOBN
        TST.L   D1
        BNE.S   JOB_NUM
NO_JOBN MOVEA.L CHAN_0,A0
        LEA.L   JOB_MES1,A1
        VECTOR  UT_MTEXT,2,CALL
        MOVE.W  (A7)+,D1
        VECTOR  UT_MINT,2,CALL
        BRA.S   JOB_RET
GOT_JOBN MOVE.L D4,D1
        ADDQ.L  #2,A7
        RTS

JOB_NME MOVE.L  A1,-(A7)
JOB_NAME MOVEQ  #0,D1
FIND_JOB MOVEQ  #0,D2
        MOVE.L  D1,D4
        QDOS    MT_JINF,1
        TST.L   D0
        BNE.S   NO_JOBK
        MOVE.L  D1,D5
        CMPI.W  #$4AFB,6(A0)
        BNE.S   NXT_JOBK
        LEA.L   8(A0),A0
        SUBA.L  A6,A0
        MOVEA.L (A7),A1
        SUBA.L  A6,A1
        MOVEQ   #1,D0
        VECTOR  UT_CSTR,2,CALL
        BEQ.S   GOT_JOBK
NXT_JOBK MOVE.L D5,D1
        BNE.S   FIND_JOB
NO_JOBK MOVEA.L (A7)+,A1
        MOVEA.L CHAN_0,A0
        VECTOR  UT_MTEXT,2,CALL
JOB_RET LEA.L   JOB_MES,A1
        VECTOR  UT_MTEXT,2,CALL
        ADDQ.L  #4,A7
        MOVEQ   #1,D0
        RTS
GOT_JOBK MOVE.L D4,D1
        ADDQ.L  #4,A7
        RTS

JOB_MES DC.W    20
        DC.B    ' is not a valid job'
        DC.B    10

JOB_MES1 DC.W   11
        DC.B    'Job number ',0

* The routine to display a file on the
* console a page at a time

MRE_RTN MOVE.L  A1,D0
        BNE.S   MORE_CMD
        LEA.L   MRE_MES,A1
        MOVEA.L CHAN_0,A0
        VECTOR  UT_MTEXT,2,CALL
MORE_QUIT MOVEQ #1,D0
        RTS
MORE_CMD MOVEA.L A1,A0
        MOVEQ   #-1,D1
        MOVEQ   #OPEN_INS,D3
        BSR     OPEN_THIS
        TST.L   D0
        BEQ.S   MORE_OPN
        MOVE.L  D0,D4
        MOVEA.L A0,A1
        MOVEA.L CHAN_0,A0
        VECTOR  UT_MTEXT,2,CALL
        MOVEQ   #'  ',D1
        MOVEQ   #-1,D3
        QDOS    IO_SBYTE,3
        MOVE.L  D4,D0
        VECTOR  UT_ERR,2,CALL
        BRA.S   MORE_QUIT
MORE_OPN MOVE.L A0,A5
        MOVEA.L CHAN_1,A0
        MOVEQ   #-1,D3
        LEA.L   D_SPACE,A1
        MOVEA.L A1,A3
        LEA.L   8(A1),A4
        QDOS    SD_CHENQ,3
NEXT_PAGE MOVE.W #3,6(A3)
        QDOS    SD_CLEAR,3
NEXT_LINE MOVEA.L A4,A1
        MOVEA.L A5,A0
        MOVE.L  #256,D2
        QDOS    IO_FLINE,3
        MOVE.L  D0,D4
        MOVEQ   #0,D2
        MOVE.W  D1,D2
        DIVU    (A3),D2
        ADDQ.W  #1,D2
        ADD.W   D2,6(A3)
        MOVE.W  D1,D2
        MOVEA.L A4,A1
        MOVEA.L CHAN_1,A0
        QDOS    IO_SSTRG,3
        TST.L   D4
        BNE.S   MORE_FERR
        MOVE.W  2(A3),D0
        SUB.W   6(A3),D0
        BPL.S   NEXT_LINE
        MOVE.W  2(A3),D2
        SUBQ.W  #1,D2
        MOVEQ   #0,D1
        QDOS    SD_POS,3
        LEA.L   MRE_MES1,A1
        VECTOR  UT_MTEXT,2,CALL
        MOVEQ   #-1,D3
        QDOS    SD_CURE,3
        QDOS    IO_FBYTE,3
        MOVE.B  D1,D2
        QDOS    SD_CURS,3
        CMPI.B  #'n',D2
        BEQ.S   END_MORE
        CMPI.B  #'N',D2
        BEQ.S   END_MORE
        CMPI.B  #escape,D2
        BNE.S   NEXT_PAGE
END_MORE MOVEQ  #0,D4
        BRA.S   LVE_MORE
MORE_FERR CMPI.L #ERR_EF,D4
        BEQ.S   END_MORE
        MOVE.L  D4,D0
        VECTOR  UT_ERR0,2,CALL
        MOVEQ   #1,D4
LVE_MORE MOVEA.L A5,A0
        QDOS    IO_CLOSE,2
        MOVE.L  D4,D0
        RTS

MRE_MES DC.W    46
        DC.B    'You must specify a'
        DC.B    ' filename with '
        DC.B    'this command',10

MRE_MES1 DC.W   6
        DC.B    'More ?'

* A routine to see if ESC has been
* pressed on channel 0. Returns with
* the zero flag set only if it has

CHK_ESC BSR.S   CHK_CHNO
        BNE.S   OUT_CHESC
        CMPI.B  #escape,D1
OUT_CHESC RTS

* A routine to check channel zero for
* input with zero timeout
* Returns with D0/flags zero if key
* pressed; byte in D1.B

CHK_CHNO MOVEA.L CHAN_0,A0
        MOVEQ   #0,D3
        QDOS    IO_FBYTE,3
        TST.L   D0
        RTS
```

NET

NEWS

Lessons of a short history

● The recent New York videotex conference provided the US comms industry with its best opportunity in a decade to assess its performance.

That performance has been at best erratic, with star operators like Dow Jones News Retrieval (built around an electronic re-packaging of the Wall St Journal) going great guns, while Florida-based Knight Ridder (sic) managed to drop around $40m in three years.

After the communal heart-searching, the Americans came to the following conclusions:

Don't imagine you can persuade domestic users to buy dedicated terminals costing hundreds of dollars, coz they won't do it. Instead, follow the path well-trodden by Compuserve, Micronet 800, etc, and flog add-ons for existing PCs.

Don't imagine you can sell an electronic Encyclopædia Britanica. The facts on a database must be individually useful, not useful only in bulk, otherwise they're too expensive to provide and therefore to buy. Talking of which . . .

Subjects for databases should have a well-defined but absorbing interest for potential users. Areas of sufficient interest have so far proved to be microcomputing (Compuserve, Micronet, etc), money-making (Dow Jones), and sex (see SEXTEX).

For the foreseeable future, boring old scrolling ASCII characters will be the format for US comms. The more exotic NAPLPS protocol hasn't taken off, despite its colour, high res, and respectable transmission rate, because of hardware costs, lack of production tools at the Information Provider end, and incompatibility with dominant US databases like Compuserve and the Source.

It's essential that Information Providers should get assistance from the network providers, because setting up in this business is just too expensive for any individual IP. Ideally, the collaboration should extend to joint ventures and linked profits – the best UK example of this approach being the Micronet/British Telecom link-up described in our last issue.

Despite all the setbacks, this is still an industry in which to make money.

Viewfax and Micronet bury the hatchet

● Details are emerging of the deal now in force between the two major Information Providers in Prestel's Microcomputing area – Micronet 800 and Viewfax 258.

In the past, relations between the two have resembled those between a racehorse and flea, with Micronet studiously ignoring its tiny rival except for occasional cries of "parasite": Micronet alleges that Viewfax has lifted its news material.

But the new arrangement is thought to involve an agreement over respective areas of operation for the two IPs, such that – for example – Micronet won't offer a service for Amstrad owners, but Viewfax will.

In addition, Viewfax has undertaken to cease competing with Micronet for new subscribers – an arrangement which relieves the minute Birmingham-based service of expensive marketing and customer service operations, while (presumably) earning it a percentage from new Micronet subs.

Just another computer application . . .

● The licensed naughtiness of America's SEXTEX network has been boosted with the launch of dedicated gay and couples-only areas.

The uncertain pleasures of talking dirty – or rather *typing* dirty – to people you've never met have long been a spin-off feature of conventional teletex services like the Source and Compuserve.

But the New York-based SEXTEX service has sought to institutionalise the wayward impulses of even the most dedicated (desicated) computer freak with a range of features including X-Mail ("Exchange lusty messages, but please don't be bashful"), Special Interests ("Sexual persuasions of all kinds are welcome") and Eroticomm ("The SEXTEX interactive facility, where several different live parties take place every night").

Gay men are offered "a contemporary, safe and uncensored method of communication", while "for the nation's swinging computer owners, SEXTEX has set aside Wednesday evenings for the benefit of couples to get to know one another".

Sounds like you? Call New York 986-5100, and you too can take part in (it says here) "the SINformation Revolution".

No movement in hackers' trial

● Robert Schifreen, erstwhile occupant of this page, and Micronet columnist Steve Gold were due for their third appearance in Bow St Magistrates Court around the time we went to press.

The two men are charged with offences under the Forgery and Counterfeiting Act after a late-night swoop on their London and Sheffield homes by a joint force of police and BT officials.

The arrests were part of a months-long police investigation following the "Duke of Edinburgh" Prestel hack late last year.

Previous Bow St appearances led to adjournments for further preparations by both sides, but insiders are hoping the magistrate might now be able to fix a date for trial.

Whatever happens, the case is unlikely to be concluded for several months.

Newsnet gives access to 2000 databased and all world news

● Busy with an energetic international recruiting drive is possibly the best of all the US databases – Newsnet.

Boasting a 4Mb daily up-date of 300 worldwide publications, Newsnet is the most powerful research tool you could wish for. Full keyword search facilities give full access to all the world's major newspapers, newsletters and business reports. And the service will do keyword searches for you on 2000 other databases, charging $35 per item retrieved.

As Newsnet points out, subs to all these publications would otherwise cost you around $50,000 a year.

Which brings us to prices. Fifteen dollars a month subs, plus a usage fee of $24/hour, gives access to an inexaustable source of top-rate info. Counting PSS charges to the States, I'd say that the average single-item survey would cost £5 or so. As you've guessed, I'm an addict.

Full gen from Newsnet at 945 Haverford Road, Bryn Mawr, Philadelphia, PA. Or phone Maralyn Hughes on 0101 215 527 8030.

Reid tells all on Micronet

● Micronet managed something of a coup recently when it cornered Acorn boss Alex Reid at a time when the whole of Fleet St was baying at his heels for some kind of a quote about his company's future.

Dr Reid appeared on the Net's Celebrity Chatline service – which allows subscribers to fire questions for real-time reply from notables inside and outside the computer industry – only days before Acorn's second shares suspension.

But the explanation may be nostalgia. Dr Reid set up Prestel and was its first boss.

DATABASE ROUTING

Denise Shamuel explains the complexities of directing the reader through a database the size of Micronet.

When picking up a magazine and flipping through its pages, the average reader won't pay much attention to the pagination, that is, the order in which articles appear. Nor indeed should they if the production editor is doing a good job. The pages of an article should appear on consecutive pages and correspond to the number shown on the contents page. All this is taken for granted. The art of pagination, and anybody in publishing will tell you that it is an art, has been built up over many years and there is no shortage of advice for those who need to know the finer points.

Magazine pagination aims to provide the reader with a logical and attractive format. Electronic media, such as Micronet, share this aim but have very different problems to overcome, that is, how to route the reader through the screens.

The sheer size of a database like Micronet creates considerable routing difficulties. Even if frames are viewed at the rate of one per second it would take over five hours to see every page. Taking the reader to the matter he or she is interested in is therefore of prime importance.

"... the shear size of Micronet's database created considerable routing problems ..."

Pyramid structures

A Micronet type database is built around the idea of a pyramid data-structure such as that represented in **Figure 1.** In this data-structure (similar to a card-index) facts are written in convenient places but not related to other items of data. The user must know where within the system the fact in which they are interested is located, and then provide the system with this address.

This is an unsatisfactory way to configure a store of data. The solution is to 'tie' related data together by way of routes that make sense to the user.

A similar structure to that of **Figure 1** is shown in **Figure 2,** but the items of data are tied together by routes. Accessing the main 'news' page will now allow a user to find out more about their main interest by selecting one of three routes – in other words by pressing a single key. The user does not now have to interrogate the database, but simply needs to press a key as directed by on-screen prompts.

This basic pyramid structure can be improved in a number of ways, but before looking at these it is worth investigating how routes are made to work.

First go back to the card index analogy. Think of a frame of data as being one card in the index. The front of the card contains the information to be stored at a particular location in the database. This information is designed for human consumption and is of little use to the computer. The back of the card is thus used to store information in a form that the computer can make use of. The 'front' page is referred to as the frame while the 'back' is a frame table.

The frame table is not normally accessible to ordinary users of the database and can only be changed by the editing system. The computer refers to any frame via the corresponding frame number. Links to other pages within the system are established by entries within the frame table.

Note that it is possible to establish a route to any page that exists within the system: it's only a matter of entering a number into the frame table. It is the skill of the database editor (or system operator) in selecting the appropriate routes that makes a system easy to read. The com-

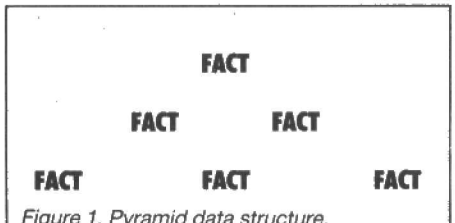


Figure 1. Pyramid data structure.

Figure 2. Linking items of data by routes.

puter has no way of knowing whether a route is appropriate – under most system editors it will not even tell an operator that a frame does not exist. For this reason, it is important that system operators spend some time as user, testing the system to see that routes work.

When starting to build up a database the first decision to be taken is where to put the data. In a book this is easy, start at the front and just keep going. An electronic book is less straightforward.

Going back to the pyramid, we gave the first frame a reference of 1. The frames leading from this frame will probably contain more detailed information about the copies introduced on frame 1.

Rather than call the second frame of data page 2, why not relate it to page 1 by including the number 1 in its page reference? Think of the resulting structure as a family tree.

With the Parents & Children system of number it is possible to make a framework of numbers and then to put data into the pages. The trick is to create a framework that is large enough to hold all the data that the system will have to hold. There is nothing more frustrating than trying to enter one last vital piece of information into a system only to find that there are no more sensible numbers to use. Rather than loose the data there is a temptation to put it on an unrelated page and promptly forget to route it properly.

The way round the problem is to put data on the grand child or even great grand child levels and to use the intermediate levels to make indices which then lead to the data. Again care should be exercised here if users are not to be led through endless frames only to discover some trivial piece of information.

"... the ideas explored here can be applied with advantage to smaller databases ..."

If possible, the idea is to estimate how many data items will be stored and then to allocate that many pages on the lowest levels. If working on your own system then it is possible that there will be many thousands of pages to choose from. But when using a third party system there may only be tens of pages at your disposal.

So far, the pyramids described resemble a triangle. In reality there is a third dimension – depth. A frame is identified by a unique number, but any page can have a number of others started up behind it. To avoid a further family tree, the frames of such a structure are tagged a,b,c . . . through to z. Thus the complete address of a frame within the database is made up of its page number (derived from its parents) and its position in the pager shown by a letter. For example, the frame

4371c

has the parents 4, 43, 437 and its position in relation to its parents is shown by its own number – 1. It is also the third frame in the page, the full components being 4371a, 4371b and 4371c.

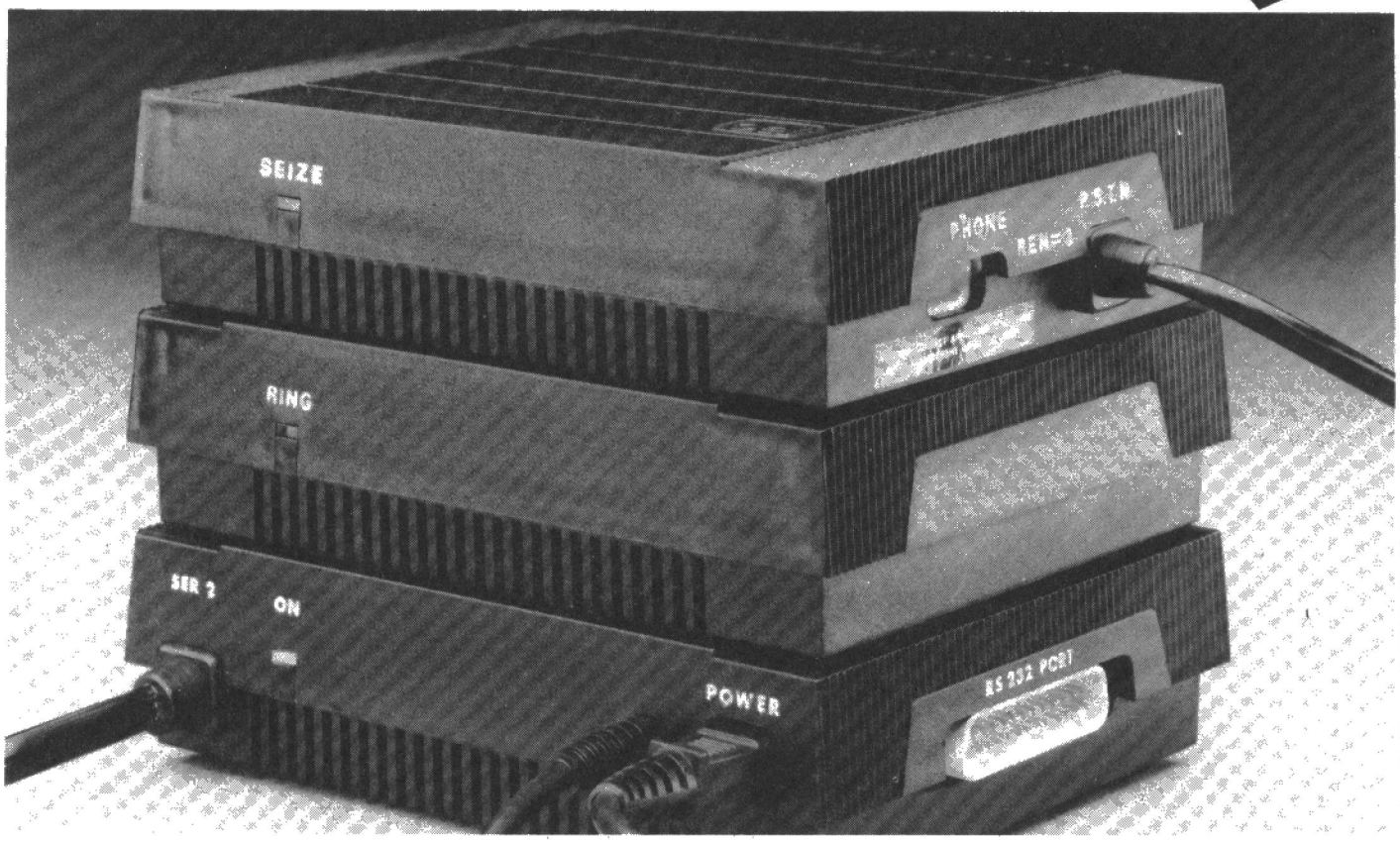
Although frames can go up to the letter z, this is a cumbersome way of thinking. Many systems (including Micronet) do not allow frames other than 'a' to be routed to, thus important information should be put onto the 'a' frame. Doing this will allow quick location. Most systems allow routing out of any frame, opening up unexpected links between ideas as they occur.

Glancing back at the pyramid containing news information, it is apparent that the most logical thing to do is to route back to the main news page from the pages at the lowest levels. But good editor will be more creative when forming the links between frames. It is unimaginative to continually refer back to a main index. Why not link the London news frame to one listing a local 'Whats On?' page?

Creating, and maintaining a Prestel style database can be a very interesting pursuit. There are a number of packages available that will allow micro users to establish such a system. Alternatively, Micronet subscribers offer pages to most users on their Gallery area.

Denise Shamuel is the database manager of Micronet. For information about the various services offered by Micronet telephone 01 278 3143.

THE ON-LINE QL



Peter Luke goes on-line with two new modems designed for the QL computer. Each offers a range of facilities but the head-to-head test reveals a clear winner in terms of performance.

QL users can now, after a considerable wait, phone home or at least their local, user friendly data base. The two modems reviewed below came to the market at roughly the same time – this though is the only thing they have in common. The Tandata modem is the product of design work undertaken by the now defunct OEL company. Considering the time taken to get the modem to the market, the final product is rather a disappointment. The Bright Star modem is a product of a very different pedigree, being the result of development work undertaken by a very small team of hardware and software designers. But the final specification is in many respects superior to that of the Tandata model.

Tandata QCOM

The QCOM modem managed to acquire a rather high profile long before it came to the market. This was the modem that OEL was working on when it ran into its much publicised financial problems some months ago. At that time, it was thought that as this was the modem that Sinclair had decided to make the official QL product, Sir Clive would step in to rescue the QCOM design. This though was before Sinclair ran into some cash flow problems of its own. In the event it was Tandata who acquired the rights to the modem.

The QCOM modem consists of three separate units that stack on top of one another to form the complete communications system. The base unit is designated Q-Connect and in addition to providing the connections to the rest of the modem system, it provides an RS232 output port via a standard 25 way plug. The middle unit of the stack is the optional auto dial/auto answer unit (Q-Call) while the top most component is the Q-Mod box, this contains the main components of the modem.

Connecting the QCOM system to the QL is a straightforward operation. First the three units are stacked together, connections between the three sections of the modem are by way of plug/socket arrangements on the top and underside faces of the various boxes – assembly is rather like building up a tower of building blocks. With the tower of building blocks. With the tower of units placed to the right of the QL, short

Auto-dialling and DTMF

During the course of reviewing the QCOM and Bright Star modems a slight problem in connection with their use on some modern telephone equipment came to light. The problem concerns PABX systems based on the DTMF signalling system – this is unlikely to be encountered in the home but could cause some business users a problem.

The difficulty is that auto-dial units operate a 'line pulse dialling' system that is required to interface to standard BT exchange lines. Many PABX systems use DTMF (Dual Tone Multi Frequency) systems for control of the exchange extensions. Such systems will ignore the attentions of an auto-dial unit. The answer is to arrange for a dedicated exchange line for use with the modem unit – this is probably a good idea anyway as it provides the most 'intimate' connection between the modem and the PSTN.

leads mean that there is little choice over where the modem is sited, the power lead of the computer is removed and plugged into the power-in socket of the Q-Con base. A lead from this unit is then plugged into the QL's power socket. Next the lead marked PSTN (Public Switched Telephone Network – the posh name of our 'phone network) is plugged into one of the new style BT phone sockets. If required a telephone terminated with the appropriate plug may be connected to the socket marked 'phone on Q Con (the fact that Q-Call, if fitted, provides both auto-answer and auto-dial options means that there is in fact no need to provide a conventional 'phone).

A series of LEDs, one per unit, indicate the status of the system. Q-Con features a LED that indicates that power is applied to the modem. The LED on the Q-Call unit is labled 'Ring'; this indicates that the auto-dial process is in operation, while the 'Seize' indicator of the Q-Mod unit signals that the modem has established communication with the remote computer's modem port.

Having completed the installation of the hardware of the modem, the next step is to load the QCOM software supplied.

The first thing to greet the user when the QCOM software has been loaded into the QL is a frame concerned with the initialisation of the software real time clock that is provided. This facility is particularly useful as it allows users to keep track of the duration for which they are logged onto any system. While cheap rate calls are cheap it is all too easy to run up large telephone bills if some watch is not kept on the length of calls.

After entering initial time and date information the system will ask the user for identification. Having done so then a request for filename prompt will appear. This filename refers to the phonebook file that allows the user to create a customised list of 'phone numbers that can be used in conjunction with the auto-dial unit to simplify the process of establishing calls. If no 'phone book file has been created, the user can opt to use the default file supplied with the QCON software.

The next menu displayed by the system is the main menu through which the other facilities of the system may be accessed. The software supports a range of operating modes although the hardware only supports the 1200/75 Prestel standard. This limits the number of menu choices that have any operational significance. In view of the comprehensive nature of the software the limited capability of the Q-MOD unit is a disappointment.

Bright Star

The Bright Star modem consists of a single box, none of the Tandata designer tower look, and in contrast to the Tandata unit has its own internal power supply. The modem thus makes no demands on the QL's power supply. Another added bonus with the Bright Star is the fact that the designers have also provided a serial to Centronics interface complete with a 2K buffer.

Connecting the modem to the QL is a straightforward process, the modem is simply connected in series with an existing 'phone circuit. The modem does not feature any front panel switches: the many functions of the unit are under complete software control. The three LEDs mounted on the front panel indicate power, on-line and carrier detect status.

The modem provides a self-test facility which allows correct operation of the modem to be confirmed. The self-test function is activated by sending the modem a DLE (Data Link Escape) character followed by CTL E. The DLE character signals to the Bright Star that the character that follows is not text but is to be interpreted as a command code. The ASCII code for DLE is generated by pressing CTL P on the QL.

Assuming that the modem is operating correctly, the self-test routine will cause the unit to 'answer back' with its version number and current status.

The Bright Star supports a real time clock and as with the Tandata modem, setting this is the first task once the system software has been loaded into the QL.

The Bright Star modem provides an auto scanning facility that will automatically select the correct transmission speed when a user logs onto a system. Sorting out comms protocols can be a difficult task even for experienced comms users – the auto-scan feature will thus be very welcome to users new to modems.

The range of facilities offered by the system are accessed by the QL's function keys – these features include toggling the printer buffer's status (on/off), selecting either 40 or 80 column screen and the quit function which returns the user to Super-Basic.

The Bright Star also features an advanced 'user-to-user' mode which makes use of some sophisticated techniques to provide reliable transfer of data at 1200/1200 baud.

The unit supplied for review was not fitted with an auto-answer/auto-dial unit but this feature will be available on production units.

The Bright Star, unlike the Tandata modem supports a full range of operational modes and coupled with the fact that it offers a wide range of additional functions, on a price performance basis it must score over the Tandata tower.

Calling all RENs

Another point to bear in mind when connecting a modem to an existing 'phone system whether at home or in a business is the total REN number of the various equipment connected to an exchange line. The total REN (Ring Equivalence Number) of the equipment connected to a single line must not exceed 4 – if it does then there is a danger that one or more of the telecom items may fail to respond to an incoming ring signal – phones will not ring, auto-answer phones will not answer and answering machines will remain inactive.

The standard BT 'phone has a REN of 1 and so four of these may be connected to various extensions throughout a home. Many other items of equipment have REN numbers that are higher than this however. Numbers of 1.5 or even 2 are not uncommon. This means that by the time a couple of 'phones and a modem have been connected to the system with say, an answer-phone machine, this may well exceed the maximum REN rating of four. If it is necessary to have a number of pieces of equipment connected simultaneously the solution is to contact the local BT sales office. They can arrange for a 'booster' unit to be fitted – this will boost the ring current and so overcome the restriction of the REN limit.

DIY ROM DESIGNER

Mike Williams continues his description of the techniques used to create sideways ROM software for the BBC micro. This month he describes how to build some useful routines into a ROM.

Last month's article on writing your own ROM into sideways RAM explained a program which provides the essential framework of a ROM. The next step is to insert *commands to build a few useful routines into the ROM; and then to set vectors to point into ROM. We've also included advice on how to modify your favourite machine code programs to work in ROM, and a method of using two piggy-backed ROMs in sideways RAM at the same time.

The routines

The first *command to be implemented enables you to store function-key definitions in ROM (or sideways RAM) and to download them into the key buffer by means of the command *FNKEYS or *FN.

There are three steps to follow in adding a *command to a ROM:

- Insert the command name in the command-table.
- Insert the command name – perhaps together with syntax – into the help-table.
- Add the code for the routine, either at the command entry point if the code is short, or further up in memory. In the latter case a JSR is inserted at the command entry point.

So, into the ROM Designer program from last month make the following changes:

- Line 2500 substitute for FNequs ("COMMAND1") – FNequs ("FNKEYS")
- Line 2610 substitute PROCstring ("FNKEYS")
- Since the required code to implement *FNKEYS is fairly short it can be inserted at the command-1 entry point. So add the following lines:

```
1862  LDX #0
1864  .keys
1866  LDA &9F00,X
1868  STA &B00,X
1870  INX
1872  BNE keys
```

The program should now be RUN.

Next you need a file of function-key definitions. Having defined the keys *SAVE the file:

eg *SAVE Keys B00 BFF

The *LOAD Keys 9F00 will load the key definitions into your sideways RAM. Press BREAK so that the system knows your ROM is there. *FNKEYS should now download the definitions and program the function-keys.

It is now easy to store many different sets of definitions and to call them up using, for example:

*AKEYS
*BKEYS

and so on. Users of Wordwise could call up their Wordwise function-key definitions with the command *WWKEYS. Just follow the steps above.

The next example is a *command to print useful information on the screen. It's called *BIRTHDAYS because it could be used to store your family's birth dates, anniversary dates etc. Again we follow the three steps:

- 2510 Substitute FNequs("BIRTHDAYS")
- 2620 PROCstring("BIRTHDAYS")

The routine is a bit longer than the FNKEY example so it is better placed further up in ROM. The program given in **Listing 1** places the routine at &9000. So for now we must insert the following:

```
1900  JSR &9000
```

Now RUN the ROM Designer (after saving the program as usual).

Listing 2 should now be entered. The program is well REM'd and can easily be modified to include your own information. The only limitation is that not more than 256 bytes of information can be displayed unless the routine is modified. After running the program your ROM should respond to *BIRTHDAYS (or *BI. for short) with the information you have placed in the DATA statements.

Coping with vectors

The third routine is *CAPS ON. Once this command has been given, then any time that Carriage Return is pressed the computer automatically switches on the Caps Lock. This avoids the irritating error messages that come from entering 'list' or 'run'. It is especially useful after entering assembler labels. The facility can be switched off by entering *CAPS OFF.

The idea is to intercept the 'insert into buffer' vector so that it points to the routine in ROM. It checks for Carriage Return and switches on the Caps Lock whenever it finds it.

Repeat the stage above.

- 2520 FNequs ("CAPS")
- 2630 PROCstring("CAPS ON/OFF")
- 1930 JSR &9400

Listing 1 has the routine to be assembled at &9400 which will implement the CAPS command. As with Listing 1 there should be enough REMs for its operation to be easily followed. However FNvector needs some explaining.

Vectors themselves are described in the Advanced User Guide. The 26 vectors are

stored in a table starting at &200. Usually, redirecting a vector to point to some routine is a simple matter of replacing the contents of the table with the address of the routine.

This is fine for addresses below &8000, but in pointing to ROM the procedure is more complicated. Fortunately FN vector makes it easy. It requires the address of the routine in ROM ('check' in this case) and the vector number (from 0 to 25) and it does all of the work.

So far your ROM should do the following:

Respond to *HELP, by printing its name.
Respond to *HELP Name, by listing the available commands.
Respond to three *commands.

Writing your own routines

Routines like screen dumps, large letters, improved trace, can all go into ROM. But, there are some points to remember.

Firstly, routines operate by actually modifying themselves as they run. Although this can work in sideways RAM it certainly won't in ROM. Since you might want to blow your successful programs into EPROM, it is better not to use the sideways RAM as RAM except where necessary, for example as a printer buffer.

Secondly, most programs do need some RAM as temporary storage or for indirect addressing. Page zero is especially important and is heavily used both by BASIC and by other ROMs. Here are some useful areas:

&70 – &8F

This is very valuable space heavily relied upon by utility programs and by other ROMs. You should try to save the values (on the stack for example) of any of these locations which you use and restore them at exist-ROM.

&3A0 – &3A6

As the Advanced User Guide points out, these locations are not used by OS1.2. But they are very popular among ROM writers as flags, so that the ROM can "remember" whether or not it has set vectors. We used &3A5 as a flag in the Caps routine to indicate that the vectors had been changed. Had you typed "CAPS OFF" when they were already off and the routine had then restored the vectors, it would probably have restored the wrong ones. Similar flags are set by ROMs with error trapping, single key, special tabs and so on. There is no reason why you should not use these locations too, but check that none of the other ROMs in your machine need them. You can do this by periodically printing out their value and seeing if they change.

&700 – &7FF

This is the input buffer space. If your routine needs some memory only for the duration of the routine then the top end of this space is pretty safe. Its a good area to use for osword and oscli parameter blocks.

LISTING 1. Caps on routine.

```
100 REM **********************
110 REM *  CAPS On when      *
120 REM *  Carriage Return   *
130 REM *     pressed.       *
140 REM **********************
150 :
160 bufvec=&22A              :REM Insert into buffer vector
170 oldbufvec=&232           :REM spare vector
180 on=&3A5                  :REM 'on' flag
190 osbyte=&FFF4
200 code=&9400               :REM modify if necessary.
210 PROCass
220 END
230 :
240 DEFPROCass
250 FOR PASS=0 TO 3 STEP 3
260   P%=code
270   [ OPT PASS
280   .init          \ check if ON or OFF
290   INY:INY        \ skip two space
300   LDA (&F2),Y    \ get next character
310   CMP #ASC"N"    \ only need check for 'N'
320   BNE caps_off
330   :
340   .caps_on
350   LDA on
360   CMP #&55       \ is routine already on?
370   BEQ end        \ if so, leave
380   LDA bufvec     \ no, so change vectors
390   STA oldbufvec
400   LDA bufvec+1
410   STA oldbufvec+1
420   OPT FNvector(check,21)
430   LDA #&55       \ set flag
440   STA on
450   .end RTS
460   :
470   .caps_off
480   LDA on
490   CMP #&55
500   BNE end        \ not 'on' so leave
510   LDA oldbufvec\ replace vectors
520   STA bufvec
530   LDA oldbufvec+1
540   STA bufvec+1
550   LDA #0         \ clear flag
560   STA on
570   RTS
580   :
590   .check         \ when 'on' the vector
600   PHP            \ point here
610   CMP #&0D
620   BNE no_change          \ not carriage return
630   PHA:TXA:PHA:TYA:PHA
640   LDA #202:LDX #32
650   JSR osbyte     \ Caps on
660   PLA:TAY:PLA:TAX:PLA
670   .no_change
680   PLP
690   JMP (oldbufvec)        \ exit via original vector
700   ]
710   NEXT PASS
720 ENDPROC
730 :
740 DEF FNvector(addr,n)
750 [OPT PASS                \ addr is Routine address
760 LDA #3*n                 \ n is vector number
770 STA &200+2*n
780 LDA #&FF
790 STA &201+2*n
800 LDA #addr MOD 256
810 STA &D9F+3*n
820 LDA #addr DIV 256
830 STA &DA0+3*n
840 LDA &F4
850 STA &DA1+3*n
860 ]
870 =PASS
880 :
```

Piggy backing ROMs

Suppose you already have an 8K ROM running in Sideways-RAM. That leaves 8K of empty space from &A000 to &BFFF. Why not put your ROM up there? Change the assembly addresses in your routines so that they are 8K up. eg FNKEYS are now at &BF00, CAPS is now at &B400.

The system now has to come to the new ROM at &A000 before that at &B000. The trick comes from modifying the two bytes at &8004,5. Remember that they normally point to the start of the ROM (entry% in our ROM): so the first step is to change them so that they point to the new ROM. Make the following modifications:

1. Note the contents of the address in locations &8004 and &8005. Change &B004 to &03, change &8005 to &A0.
2. Make ROM start in the ROM Designer &A000 and change the commands so that they call the new routine addresses.
3. Change line 760 from BTS to JMP address from 1 above.
4. Run the ROM Designer so that your ROM is assembled at &A000.

Now with a bit of luck you should have two ROMs where before there was only one.

LISTING 2. ★BIRTHDAYS.

```
100 REM ****************
110 REM *  BIRTHDAYS   *
120 REM *     or       *
130 REM *   whatever   *
140 REM ****************
150 :
160 code%=&9000 : REM can be modified
170 osnewl=&FFE7
180 osbyte=&FFF4
190 oswrch=&FFEE
200 PROCassemble
210 END
220 :
230 DEFPROCassemble
240 FOR PASS=0 TO 3 STEP 3
250   P%=code%
260   [ OPT PASS
270   LDY #0
280   .newline
290   JSR osnewl
300   LDA #32:JSR oswrch:JSR oswrch:
310   .print_line
320   JSR oswrch
330   INY
340   LDA info_table,Y
350   BEQ newline  \ each line ends with '0'
360   CMP #9       \ end of info marker
370   BNE print_line
380   JSR osnewl
390   RTS
400   \
410   .info_table BRK
420   ]
430   NEXT PASS
440 :
450 rom%=P%
460  REM Now poke in the data
470 REPEAT
480    READ A$:IF A$<>"END" THEN PROCstring(A$)
490    UNTIL A$="END"
500 ?rom%=9
510 ENDPROC
520 :
530 DEFPROCstring(A$)
540 $rom%=A$:rom%=rom%+LENA$
550 ?(rom%)=0:rom%=rom%+1
560 ENDPROC
570 :
580 REM ** Place your own info here **
590 DATA Mike - Dec. 1st
600 DATA Nathan - Jan 24th
610 DATA Yuki - June 15th
620 DATA Momoko - July 3rd
630 DATA END
```

THE FIRST SPECTRUM WORDPROCESSOR IN FIRMWARE PART SIX

Bubble sort

Richard Sargent has included a fast bubble sort routine and special print routines to create poster sized letters in his Spectrum wordprocessor project.

More code

The bare bones of the bubble sort code are shown in **Listing 1.** It should be possible to add it to existing Z80 wordprocessors because, like the number-conversion routine, it has the virtue of leaving the document length severely alone: the sort should be invisible as far as the host wordprocessor is concerned. The listing is heavily annotated to show what's going on, but the object code isn't printed, since it would be foolhardy to type in the raw bytes and expect them to work their miracles immediately.

This is a routine which needs a certain amount of care in its use: the bubble sort code itself is about 240d bytes long, excluding checking routines, which must be created as need arises. If, for example, records containing five fields are set up, then a safety program must be written to ensure that the correct number of delimit symbols (four in this case) are present in every record, and that no null fields have appeared. A blank field containing at least one space character is allowed, but a null field of two markers next to each other is not. The check-code is easy to write but is lengthy due to the need to include informative error-messages such as "field marker absent in record 52" and "the end section-marker is missing". The checking routine must also establish where the Sort-File begins by searching for the **SECTION-MARKER,** and the start-address should be loaded into **START.** The file can then be searched from that point onwards until the second SECTIONMARKER is found, which indicates the end of the Sort-File. The total number of records present will need to be counted and placed in **RECNUM.** The variables **DEPTH** and **FIELDNUMBER** have default values of one, but different values may be supplied by the user whenever a sort is about to be performed.

This article completes the ROM-based Spectrum wordprocessor project. For details of how to obtain the software write to Spectrum Wordprocessor, E&CM, Priory Court, 30-32 Farringdon Lane, London EC1R 3AU.

Sorting is achieved by the comparison of the ASCII values of characters. "adam" will be placed lower in a list than "BOB" since "a" has a value of 97 and "B" has a value of 66. This is often undesirable, so a **CASELOCK** variable has been provided, which, when set to zero, causes the routine to ignore differences in case. A number sort, too, requires care, since a leading space is not the same as a leading zero. A field containing % 9% will be placed ahead of a field containing %01%. The solution to this problem is consistency. Either always use leading zeros or always use leading spaces. Better still, use stock numbers starting with 1000. Finally, in order to sort your records chronologically, you must enter the date in year-month-day order.

Special print routines

One of the special ROM printer output routines is called PQP (Poster-quality-print) since, in its default setting, it is capable of printing characters 17mm high by 14mm wide. This gives 15 character locations across a piece of A4 paper. What, though, of NLQ (Near letter quality) print? It is possible to get close to NLQ. Design a suitable font in RAM and the ROM routine will print it. (It does take a long time to design NLQ characters on graph-paper, and "borrowing" commercial designs will only work if they were formed on an identical matrix.)

What the PQP routine really does is drive an Epson-type printer in high-resolution bit-image mode. Each character printed is formed on a 16XB matrix, and there is sufficient room to print 56 characters across the page when the routine is set to print in **SIZE1. SIZE2** puts about 28 characters across a page and **SIZE4** is the mode which produces the large poster-like letters, 14 to the page. The routine itself is 1K and the shape table for the 96 ASCII alphanumeric characters uses a further 1.5K.

The other special printer routine is a short affair, designed specifically to enable the WYm?0 Graphic Mode of the Epson RX80 to be used easily from the wordprocessor. The RX80 has 32 built-in graphics which are very useful and, unlike the graphics of bit-image mode, they are printed at the normal speed of 100 CPS.

LISTING 1. Bubble sort.

```
SCRATCHRAM  EQU 23296
SECTIONMARKER EQU 40H       ;limit marker
FIELDMARKER EQU 25H         ;field marker

          ORG 7E00H

START     DW 0              ;to be set by user
RECNUM    DW 1              ;to be set by user
DEPTH     DB 1              ;to be set by user
FIELDNUMBER DB 1            ;to be set by user
CASELOCK  DB 0              ;0=sort will ignore case
COUNT     DW 0
SWAPLATCH DB 0
LATCH     DB 0
TEMPSTORE DW 0

;ENTER WITH THE CONTENTS OF
;(DEPTH), (FIELDNUMBER), (START) and (RECNUM)
;ALL VALID

SORT   XOR A              ;
       LD (SWAPLATCH),A ;Empty SWOP latch

       LD HL,(RECNUM)     ;
       DEC HL             ;
       LD (COUNT),HL      ;Set up records counter

       LD HL,(START)      ;Initialise HL

SORT2  CALL FINDFIELD     ;Points HL to desired
       PUSH HL            ;field (record A) and save it
       CALL NEXT_CR       ;Advance to end-of-record
       CALL FINDFIELD     ;Points HL to desired
       EX DE,HL           ;field (record B) and put in DE
       POP HL             ;Recover one pointer and
       LD (TEMPSTORE),DE;save the other one

       CALL COMPARE
       JR C SORT3         ;if CARRY do a swap
       JR Z SORT4         ;if ZERO don't swap

SORT3  CALL LAST_CR       ;Adjust HL pointer
       INC HL             ;and then swap
       CALL SWAP          ;the 2 complete records

SORT4  LD BC,(COUNT)      ;See
       DEC BC             ;if we
       LD (COUNT),BC      ;have
       XOR A              ;"rippled"
       OR B               ;through
       OR C               ;all the
       JR Z SORT5         ;records yet

       LD HL,(TEMPSTORE);If not
       LD A,(HL)          ;adjust
       CP 0DH             ;the HL
       JR Z SORT2         ;pointer
       CALL LAST_CR       ;and
       JR SORT2           ;keep trying!

;A pass through all the records has been
;made, but unless SWAPLATCH is zero the
;whole process must be repeated,,,

SORT5  LD A,(SWAPLATCH)
       OR A
       JR NZ SORT
       RET
SWAP   PUSH HL
       LD DE,SCRATCHRAM
       ;copy record A to scratchram
       CALL MOVE
       ;HL now pointing to where record A was
       POP DE
       ;copy record B to that position
       CALL MOVE
       LD HL,SCRATCHRAM
       ;and copy scratchram contents to the
       ;position following
       CALL MOVE
       EX DE,HL
       DEC HL
       ;HL has advanced through the file
       LD A,1
       LD (SWAPLATCH),A
       ;and the swap is registered
       RET
MOVE   LD A,(HL)
       LD (DE),A
       INC HL
       INC DE
       CP 0DH
       JR NZ MOVE
       RET

COMPARE   LD A,(DEPTH)      ;The depth of sort
          LD B,A            ;controls the comparison loop

          LD A,1            ;Keep going while
          LD (LATCH),A      ;LATCH=1

COMP2     LD A,(LATCH)      ;
          OR A              ;
          RET Z             ;Possible exit No 1

          INC HL            ;
          INC DE            ;Step on pointers

          ;a field in record A
          LD A,(HL)         ;
          CP FIELDMARKER    ;
          RET Z             ;Possible exit No 2
          CP 0DH            ;
          RET Z             ;Possible exit No 3

          ;a field in record B
          LD A,(DE)         ;
          CP FIELDMARKER    ;
          RET Z             ;Possible exit No 4
          CP 0DH            ;
          RET Z             ;Possible exit No 5

          ;the character is valid
          PUSH AF           ;Save character  Record B
          LD A,(CASELOCK)   ;
          OR A              ;Jump to COMP3 if
          JR Z COMP3        ;ignoring case
          POP AF            ;Reclaim character  Record B
          CP (HL)           ;Compare it with char from A
          JR COMP4
          ;Conversion to upper case,,,
COMP3     POP AF
          CALL UPPER
          PUSH AF
          LD A,(HL)
          CALL UPPER
          LD C,A
          POP AF
          ;,,,and do the comparison
          CP C
COMP4     RET C             ;Exit 6 : a swap is required
          JR Z COMP5        ;The characters match
          ;A swap is not required
          ;The characters do not match
          ;therefore there is no need to
          ;compare further
          XOR A             ;
          LD (LATCH),A      ;so set LATCH
COMP5     DJNZ COMP2        ;Go round again
          ;or fall through if complete
          ;depth reached
          XOR A
          RET ;and thus escape on exit 7

;POINTER-SETTING ROUTINES

FINDFIELD PUSH BC
          LD A,(FIELDNUMBER)
          LD B,A
FD1       DJNZ FD2
          POP BC
          RET

FD2       INC HL
          LD A,(HL)
          CP FIELDMARKER
          JR NZ FD2
          JR FD1

NEXT_CR   LD A,0DH
SW2       INC HL
          CP (HL)
          JR NZ SW2
          RET

LAST_CR   LD A,0DH
SW1       DEC HL
          CP (HL)
          JR NZ SW1
          RET

;CASE-CONVERSION ROUTINE

UPPER     CP "a"
          RET C
          CP "z"+1
          RET NC
          OR A
          SBC A,20H
          RET
```

More This Month at Maplin

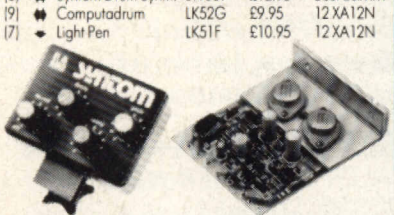
Colour–coded IDC cables
16-way (XR80B) ONLY 32p per metre.
20-way (XR81C) ONLY 40p per metre.
26-way (XR82D) ONLY 54p per metre.
34-way (XR83E) ONLY 70p per metre.
40-way (XR84F) ONLY 82p per metre.
50-way (XR85G) ONLY 99p per metre.
4-way flat flexible telephone lead (XR86T) ONLY 18p per metre.
Stepper motor 48 steps/rev, 12V 0.13A per phase, 4-phase unipolar, 57g, working torque 8mNm max. ONLY £9.95 (FT73Q).
Driver chip for motor: SAA1027 ONLY £3.75 (QY76H).
★SAVE★ 1 Kit containing everything you need: motor, SAA1027, data sheet and passives ONLY £13.35 (LK76H).

Sounds Terrific

Professional Quality
High Power Loudspeakers
featuring:
★ Virtually indestructible high-temperature voice-coil reinforced with glass-fibre.
★ 100% heat overload tolerance.
★ Advanced technology magnet system.
★ Rigid cast alloy chassis.
★ Linen or Plastiflex elastomer surrounds.
★ 5-year guarantee (in addition to statutory rights).
Prices from £18.95.
Send S.A.E. for our free leaflet XH62S.

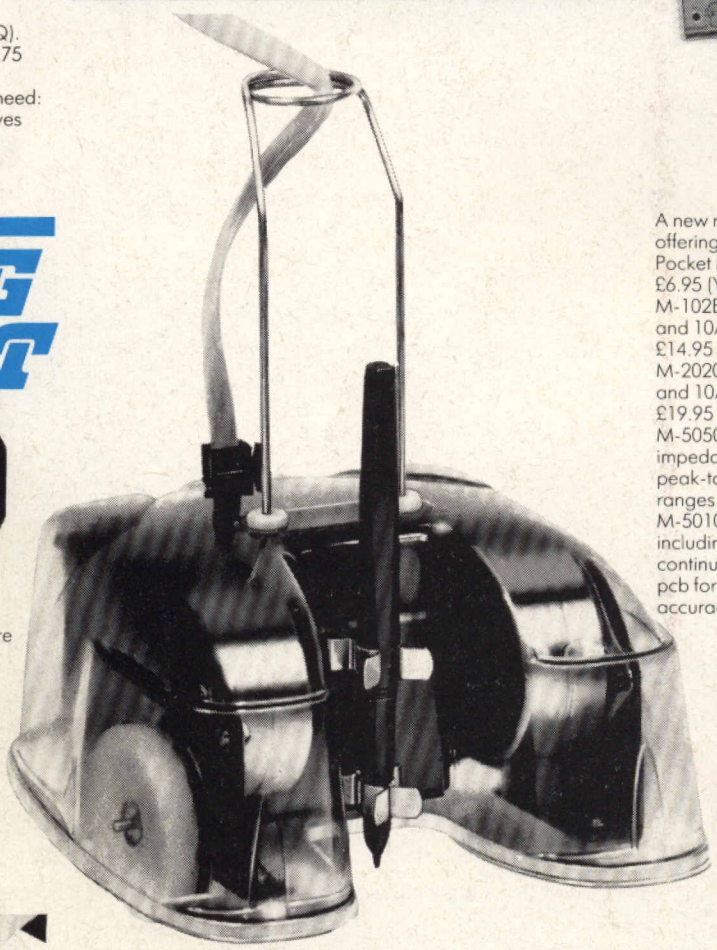
Top Ten Kits

THIS/LAST MONTH		DESCRIPTION	CODE	PRICE	BOOK
1.	(1)	Live-Wire Detector	LK63T	£2.95	14 XA14Q
2.	(2)	75W Mosfet Amp.	LW51F	£15.95	Best E&MM
3.	(3)	Car Burglar Alarm	LW78K	£7.49	4 XA04E
4.	(4)	Partylite	LW93B	£10.95	Best E&MM
5.	(5)	U/sonic Intrudr Dtctr	LW83E	£10.95	4 XA04E
6.	(6)	8W Amplifier	LW36P	£4.95	Catalogue
7.	(10)	Logic Probe	LK13P	£10.95	8 XA08J
8.	(8)	Syntom Drum Synth.	LW86T	£12.95	Best E&MM
9.	(9)	Computadrum	LK52G	£9.95	12 XA12N
10.	(7)	Light Pen	LK51F	£10.95	12 XA12N



Over 100 other kits also available. All kits supplied with instructions. The descriptions above are necessarily short. Please ensure you know exactly what the kit is and what it comprises before ordering, by checking the appropriate Project Book mentioned in the list above.

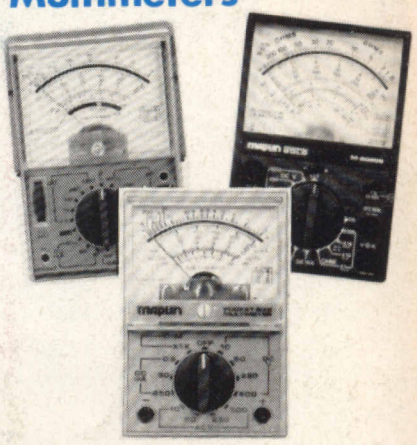
Is it a turtle? Is it a robot? Is it a buggy? Yes! it's Zero 2.



- May be used by any computer with RS232 facility.
- Stepper Motor controlled.
- Half millimetre/half degree resolution.
- Uses ordinary felt-tip pens.
- Built-in 2-tone horn, line-follower. LED indicators.

The Zero 2 Robot is the first truly micro robotic system available and remarkably it costs less than £80. Complete kit (only mechanical construction required) £79.95 (LK66W).
Full details of power supply and simple interfacing for BBC, Commodore 64 and Spectrum, in Maplin Magazine 15 price 75p (XA15R).

More Choice In Multimeters



A new range of very high quality multimeters offering truly amazing quality at the price.
Pocket Multimeter, 16 ranges, 2,000Ω/V DC/AC £6.95 (YJ06G)
M-102BZ with continuity buzzer, battery tester and 10A DC range, 23 ranges, 20,000Ω/V DC £14.95 (YJ07H)
M-2020S with transistor, diode and LED tester and 10A DC range, 27 ranges, 20,000Ω/V DC £19.95 (YJ08J)
M-5050E Electronic Multimeter with very high impedance FET input, 53 ranges, including peak-to-peak AC, centre-zero and 12A AC/DC ranges £34.95 (YJ09K)
M-5010 Digital Multimeter with 31 ranges including 20Ω and 20μA DC/AC FSD ranges, continuity buzzer, diode test, and gold-plated pcb for long-term reliability and consistent high accuracy (0.25% +1 digit DCV) £42.50 (YJ10L)

The Maplin Service

All in-stock goods despatched same day for all orders received before 2.00 pm.
All our prices include VAT and carriage (first class up to 750g).
A 50p handling charge must be added if your total order is less than £5.00 on mail-order (except catalogue).

AMERICAN EXPRESS | map | Access | VISA

☎ Phone before 2.00 p.m. for same day despatch.

MAPLIN

MAPLIN ELECTRONIC SUPPLIES LTD
Mail-order: P.O. Box 3, Rayleigh, Essex SS6 8LR.
Telephone: Southend (0702) 552911
SHOPS
- BIRMINGHAM Lynton Square, Perry Barr, Tel: 021-356-7292.
- LONDON 159-161 King Street, Hammersmith, W6.
Telephone: 01-748 0926.
- MANCHESTER 8 Oxford Road, Tel: 061-236 0281.
- SOUTHAMPTON 46-48 Bevois Valley Road. Tel: 0703-225831
- SOUTHEND 282-284 London Rd, Westcliff-on-Sea, Essex.
Telephone: 0702-554000. *Shops closed all day Monday.*

1985 CATALOGUE

Pick up a copy now at a branch of W.H. Smith* or in one of our shops. Price £1.35, or by post £1.75 from our Rayleigh address (quote CA02C).
**Some branches are now out of stock.*

All offers subject to availability.
Prices firm until 9th November 1985.